WO03021416

Publication Title:

METHOD AND APPARATUS FOR OBJECT ORIENTED MULTIMEDIA EDITING

Abstract:

Abstract of WO03021416

A computer editing system for editing and creating a multimedia program in a visual workspace (1), which provides a plurality of segments of audio or video data, displays an icon (10) representing each of the segments, associates a first one of the segments (2) with at least one other of the segments, displays an indication of the association of the first to the other of the segments, thereby composing a program of the segments, wherein the program may be played in a sequence determined by a user in accordance with the associations.

Data supplied from the esp@cenet database - Worldwide b49

Courtesy of http://v3.espacenet.com

This Patent PDF Generated by Patent Fetcher(R), a service of Stroke of Color, Inc.

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
13 March 2003 (13.03.2003)

PCT

(10) International Publication Number
WO 03/021416 A1

(51) International Patent Classification[7]: G06F 3/00, 15/82

(21) International Application Number: PCT/US02/27820

(22) International Filing Date: 30 August 2002 (30.08.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/946,831 4 September 2001 (04.09.2001) US

(71) Applicant: COMMAND AUDIO CORPORATION [US/US]; 101 Redwood Shores Parkway, Suite 100, Redwood City, CA 94055 (US).

(72) Inventors: LINDEN, Thomas; 20420 Idylwild Drive, Los Gatos, CA 95033 (US). SWERDLOW, Serge; 725 Cowper Street, #25, Palo Alto, CA 94301 (US). CHERNIKOV, Andrey; 20 Dolores Way, Orinda, CA 94563 (US).

(74) Agents: ALLENBY, Christopher, B. et al.; Skjerven Morrill LLP, 25 Metro Drive, Suite 700, San Jose, CA 95110 (US).

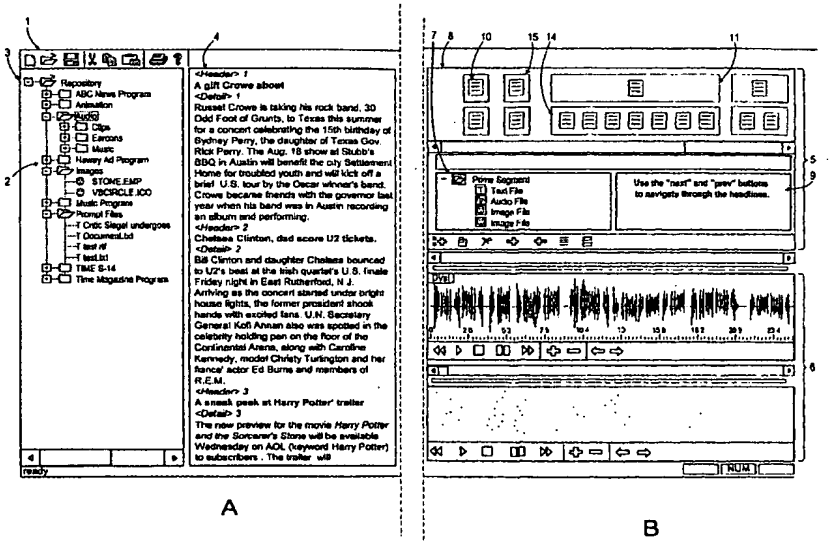
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

[Continued on next page]

(54) Title: METHOD AND APPARATUS FOR OBJECT ORIENTED MULTIMEDIA EDITING



(57) Abstract: A computer editing system for editing and creating a multimedia program in a visual workspace (1), which provides a plurality of segments of audio or video data, displays an icon (10) representing each of the segments, associates a first one of the segments (2) with at least one other of the segments, displays an indication of the association of the first to the other of the segments, thereby composing a program of the segments, wherein the program may be played in a sequence determined by a user in accordance with the associations.

WO 03/021416 A1

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHOD AND APPARATUS FOR OBJECT ORIENTED MULTIMEDIA EDITING

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever.

A computer program listing Appendix that includes a software program code executable on computer as described below is part of this application.

FIELD OF INVENTION

The present invention relates to a method and apparatus for editing and creating a multimedia program in a visual workspace.

BACKGROUND

Computer-based editing systems for video and audio information are well known, and a number of such systems are commercially available. These typically allow the user of the editing (or production) system, who is referred to as an editor or producer, to create and edit programs from segments or clips of video or audio, and which usually also include text and graphics material.

In addition, it is known to include what are called links or hyperlinks in audio and/or video multimedia presentations. These links allow the viewer or listener of the program to jump from one program segment to another at his/her election rather than to merely listen or watch the program in a conventional sequential order. Such links are well known in the computer context, but are not so limited.

For instance, see U.S. Patent Application No. 09/320,132, filed May 25, 1999, entitled "Playing Audio Of One Kind In Response To User Action While Playing Audio of Another Kind", filed by William J. Loewenthal et al., commonly owned with this application; and also see commonly owned U.S. Application No. 09,272,633, filed March 18, 1999, entitled "Program Links and Bulletins for Audio Information Device", filed by William J. Loewenthal et al., all incorporated herein by reference in their entireties. The above-referenced applications disclose use of links between various data files which are programs, or parts of programs, which advantageously allow the user to traverse from one data file to another data file depending in some embodiments upon predetermined context related relationships. This is analogous to the

hyperlinks well known in the computer field in, for instance, websites. Here it is used in the context of a radio-type audio receiver, and is also adaptable for a television-type receiver ("Player"). Of course, this requires that the receiver include local storage for storing the program contents so that the user can navigate between the program segments. Also, see U.S. Patents No. 5,406,626; 5,524,051; 5,590,195, all issued to John O. Ryan, and incorporated herein by reference in their entireties.

Hence, while such linked audio programs are known and also, in general, computer editing of video and audio programs is known, in the prior art construction of such linked programs requires substantial effort even using computer based editing.

SUMMARY OF THE INVENTION

The present system allows the producer to more easily produce the programs by allowing him/her to make the associations ("links") between and among program segments in a visual workspace. The program produced is thereby not unidirectional (sequential in time only). The system allows the producer to edit the programs into a program that does not playback from beginning to end in a unidirectional (time) line, but rather, a program that has associations that allow playback of the produced programs in an order that is determined by the listener (i.e., the programs are interactive). Unidirectional editing as described above is known in the prior art (see e.g., U.S. Patent No. 5,892,507, Moorby et al., incorporated by reference in its entirety). This is a type of editing in which multimedia data can be linked together to be played back in the form of a story from beginning to end along a unidirectional (time) line.

The present system is an improvement on this time-based editing process that producers have used in the past. The present system provides a visual workspace, on a single screen, that enables the producer to isolate the segments comprising any program, to convert these segments into objects, to establish non-linear (multi-dimensional) relationships ("links") between and among any number of such objects, and to prepare multimedia programs. The benefits of the system over the well-known, time-based editing process include the production of programs more quickly, the reduced need for technical expertise on the part of producers, and the improvement in the quality of programs.

Hence, there is provided here a system suitable for creating and editing such complex linked programs using a graphical user interface which allows easy linking by ordering icons depicting the various segments on a screen, and providing on the screen a spatial indication of same.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates a sample screen of the visual workspace.

Figure 2 illustrates a linking of objects coded with the markup language.

Figure 3 illustrates a sample view of the composer of FIG. 1, containing header and detail content elements.

Figure 4 illustrates a sample of a "Player" interface on a general purpose computer that can be used in conjunction with this invention to play the produced program.

Figure 5 illustrates a flowchart of the production of a program using the present system.

DETAILED DESCRIPTION

According to an embodiment of the present invention, multimedia programs are created and edited (collectively "produced") using computer software that allows multimedia content (e.g., audio, video, text, graphics) to be displayed as objects on a screen associated with a computer. A simple-to-use visual workspace (graphical user interface) is used to produce programs with complex associations. The software runs on, e.g., a standard computer executing the Windows 2000 operating system and is coded, e.g., in the C++ computer language.

Figure 1 shows a sample screen used interactively by the producer (editor), who is a person. This screen 1 is displayed by the software executed on the computer. Coding this software in a suitable language is well within the skill of one of ordinary skill in the art in light of this disclosure. In one embodiment the software is an add-on module to the commercially available Multitrack Editor V3.98, part of the DigAS editing system available from D.A.V.I.D. GmbH of Germany. The producer views the multimedia files 2 or content in the content explorer 3. The content explorer 3 is a box on the screen that displays as icons the multimedia files 2 (program segments) available to the producer for editing. The producer can choose among available multimedia data files to produce programs. The file icons can be conventionally dragged and dropped (e.g., using a mouse associated with the computer) into the prompter 4, composer 5, and/or waveform editor 6 portions of the computer screen. The content explorer 3 contains icons for multimedia files 2, including audio, video, text and graphics files, as well as completed programs. This content (the actual program segments) could come from outside sources such as magazine articles, quiz shows, or audio/video sports and traffic updates. The content could also come from files or records in a computer database. Content is of the type available from LAN, local, and remote sources in, e.g., conventional .wav, .txt, .mp3, .jpg, and .rtf data file formats.

The composer 5, as illustrated in Figure 1, provides the producer with a visual workspace to produce programs. The composer provides a visual representation of the structural parts of the program. The composer has three viewing boxes. In these different boxes, the producer can drag and drop multimedia files for viewing or editing. First, there is a multimedia viewer 7, which displays the types of files that are contained in each segment 10. Second, there is a navigational viewer 8, where the navigational (or multidimensional) structure of the program, or the sequence of segments, is created. Third, there is a view box 9, in which the producer can view and edit the text or image files without having to open the individual files.

In the composer 5, the producer can view the segments 10 available for program composition. The composer allows the producer to view and edit segments in the navigational viewer 8. The story or content in each segment icon is labeled so that the producer can edit the program without having to click each segment icon to determine its contents. There are also conventional search mechanisms to find a segment by its label. This is especially necessary for larger programs. Each segment 10 is conventionally represented by an icon. An unlimited number of content elements 11 (rectangles) are available in the navigational view 8. When the number of content element 11 sets reaches the edge of the composer view, the producer is able to further view the content element 11 sets beyond the edge of the view by conventional scrolling. The properties of the segments 10 in the content elements 11, such as the name or length of the segments, can be viewed in a multimedia viewer 7 by suitably selecting a segment. The producer can insert, delete, and/or append segments in the composer 5. The producer can review and listen to (or view) the content layers (discussed below) within segments in the composer 5 using the view box 9 and the waveform editor 6. The producer can examine segments in any order, moving back and forth or up and down between content elements, as illustrated in Fig. 3, and as explained in detail below. The producer can also listen to or view header content elements sequentially in the composer.

The segments 10 are placed in the content elements 11. These content elements are shaped like rectangles in the composer's navigational viewer 8. The content elements 11 are empty until the producer places segments 10 within the content elements 11. These content elements are arranged in levels. Figure 3 shows the arrangement of the content elements into header content elements 12 (top level) and detail content elements 13 (lower level).

The content elements are either prime, meaning they contain only one segment, or are compound, meaning they contain more than one segment. A content element may contain any number of segments. Figure 1 illustrates examples of a compound content element 14 and a prime content element 15. When the content element contains so many segment icons that they

are no longer viewable on the screen, the producer can scroll within the content element to see the multiple icon segments in the content element on the screen at the same time. The producer creates the associations between and among these segments, which here include associations other than merely sequential in time.

As seen in Figure 3, one embodiment of the composer provides a "template" that consists of content elements for header information of a story and content elements for the details of a story. There are three types of programs that can be produced. The most basic type is a one-level program. The end user (who ultimately plays the produced program) can determine the order this program is played, scanning forward or backwards through segments. A complex type of program is a two-level program. A two-level program has (linked) headers and details which are both content elements. A header provides, e.g., a short introduction to a story. A detail provides, e.g., the body of a story. The content elements (icons) are arranged in a template in the composer to make it easier for the producer to make the associations in the program. A template can be spatially arranged to have header content elements 12 on the top level and to have detail content elements 13 on the lower level of the composer's navigational viewer 8.

Another complex type of program is a three-level program. (The number of levels is not limited.) A template could also be arranged to have a third level of detail content elements. The producer uses this third level of detail content elements to arrange segments in the program which further relate to the header or details of the story. For example, the third level contains an advertisement for the program, along with an option for the end user to make a purchase. In this example, the producer could also place a segment with a video clip in the third-level detail content element that would play once the end user made the purchase. This third level of content elements is associated with either or both of the other two levels of content elements.

The producer can via the user interface make any number of associations between and among the header content elements and detail content elements. For example, a header content element could be linked with one or more detail content elements. A producer could link these by introducing on the screen an explicit relationship graphic line between the header and each of the detail content elements. Alternatively, a producer could make a duplicate header content element (using conventional shading or coloring on the icon to indicate it is a duplicate) and copy it on the screen to a location above each of the detail content elements that the producer intends to be associated with the header content element. One detail content element could also be shared by (linked to) multiple header content elements using similar linking techniques.

The prompter 4, as shown in Figure 1, allows the producer to create text content and open an existing text file to put into a program. The prompter 4 is for viewing and editing text. Text

can be imported from a text file or word document or composed directly in the prompter 4. The prompter 4 can receive files from the content explorer 3 and from outside applications. The prompter 4 allows the producer to mark parts of the text with tags 16 (Figure 3). The producer can tag parts of text for re-evaluation later in editing process. The tags 16 will remain in the prompter 4 as long as any of these remain in text. When the text is recorded into audio (using, for example, conventional text to speech methods), the recorded audio would be automatically separated into individual segments based on the tags 16 placed in the text.

As illustrated in Figure 1, the visual workspace also contains a conventional audio waveform editor 6. The waveform editor 6 allows the producer to edit audio files and segment audio programs.

Segments of multimedia data are conventionally created by the producer. The producer places a content layer or multiple content layers of multimedia data in the segments. The producer can drag and drop multimedia data files 2 (content) from the content explorer 3 into the individual segments that make up a program. Figure 2 illustrates in a chart the content layers 18 that make up a segment (Figure 2 is for conceptual purposes and is not part of the user interface). Each content layer contains multimedia data. Content layers 18 may contain either audio, video, text or graphics data.

The editing system also enables the producer to apply differing conventional compression techniques to each content layer within each segment. Different content may need to be compressed at different rates in order to minimize bandwidth consumption in the system. For example, if an audio layer contains music and speech, an author using the editing system can apply a different compression to the music and the speech.

After the segments are edited by the producer in the visual workspace by manipulating their icons, in one embodiment the segments are coded in a markup language. This is done to provide an exportable file using the markup language (ML). MLs are routinely used in this context for exportable computer files. Figure 2 illustrates the associations that the markup language allows between the segments. The segments are coded by the ML so that they can be represented as objects 17. The objects 17 are segments, coded with the ML. After coding with the ML, the segment is an object 17, composed of content layers 18. These objects 17 contain the same audio, text, graphics or video content layers 18 that were in the original segments. This markup language establishes relationships between and among objects (using well-known hyperlinking technology). The ML also enables the graphical user interface ("GUI") on the associated "Player" to interact with the content layers of the objects. The Player is a receiver unit, as described in the above-referenced patent applications, to which an embodiment of this

invention delivers the programs edited by the editing system. The graphical user interface depicts a ML multimedia object composed of ML descriptions 19. Thus, the editing system allows the producer to assemble a markup language multimedia object from multiple source components.

Once the program is completed, the program can be played in the visual workspace on a general purpose computer using a Player application program, or can be played using the physical dedicated Player device. The computer requires the Player application to play the program. An example of a Player application program interface 20 on a general purpose computer is illustrated in Figure 4. The Player interface 20 has a small screen and various controls as illustrated. The editor or end user can listen to the header of a story and can listen to the details of that story by pressing the "more info" button 20a on the Player interface 20. For example, if the end user is listening to a story about Tiger Woods, and mention of his equipment is made in that story, the end user can press the "more info" button 20a on the Player interface and hear about the equipment Tiger Woods uses and could also hear an advertisement for this equipment and even make a purchase of the advertised equipment. The user could also simultaneously watch footage of Tiger swinging a golf club in the viewing screen 20b of the Player. If the editor or end user does not want to listen to the details of a story, he can continue listening to the default program of continuous header story segments that the producer has produced using this invention. Other default programs may be produced; the producer could, for example, produce a program that would play the header and each detail of a story even if not prompted by the user. The dedicated Player is of the type described above and has similar functionality as interface 20.

The present system allows the producer to more easily produce linked type programs by allowing him/her to make the associations between and among program segments in a visual workspace. The process of creating associations is made easier because the producer can not only arrange the stories in content elements, but can also see the segments and what they contain. Thus, for example, in a header content element, the producer can see there is a heading segment of a story about Tiger Woods winning a tournament, and in the detail content element there is a segment about the details of the story about Tiger winning a tournament. Further, the producer can see that a segment (or segments) in a third-level detail content element contains an advertisement for golf clubs in which the end user could, for example, make a purchase or watch a video of Tiger using the golf clubs being advertised.

Figure 5 illustrates a flowchart of the production of a program using the present system. First, program content or multimedia data files are stored in a local database 22 associated with

the editing computer. The producer uses these files to produce programs in the visual workspace 21. When production is complete, the programs are saved in the database 22 and notification is sent to the ML generator 23 (which is the ML described above). The ML generator 23 retrieves the programs from database 22 and codes the program with the markup language. The resulting marked-up program is saved in a new database 24 also associated with the editing computer. The distribution system 25 is then notified that the program is complete. Distribution system 25 is, e.g., a wireless broadcast system, such as the system utilized by the Player, as described in the above-referenced patent applications. Distribution system 25 can also be, e.g., an internet system, such as the system utilized by an internet based version of the Player. In addition, the distribution system 25 can be, e.g., a physical data storage media such as Compact Flash, Tape, or Compact Disk. Finally, the Player 26 receives content from the distribution system 25 and stores the content in its local database 27. The stored content is then available for interactive playback through the Player's 26 graphical user interface.

The computer program listing Appendix, which is a part of this application, includes a description including computer code of the markup language described above, and is entitled "CAML: Command Audio Markup Language Specification." The Appendix also includes computer code, and descriptions thereof in the form of comments, to carry out the system in accordance with this disclosure. That description is entitled "On-Demand Authoring (ODA) Studio Design Specification."

Having described the embodiments of the invention, it should be apparent to those skilled in the art that the foregoing is merely illustrative and not limiting, having been presented as an example only. Modifications and other embodiments are within the scope of one of ordinary skill in the art and are contemplated as being within the scope of the invention as defined by the following claims.

APPENDIX A

CAML: COMMAND AUDIO MARKUP LANGUAGE SPECIFICATION (Release 1.1)

COMMAND AUDIO CONFIDENTIAL

Document Number: 067-0020-A2
Version Number: 1.1
Document File Name: CAML Specification
Date: August 1, 2001
Authors: Ed Costello, Dmitry Kirsanov, Thom Linden, Alex Richter, Serge Swerdlow,
Formats: DOC
Location: \\xxxxxxxx\c:\xxxxxxxxx.doc
Online Location: http://xxxxx/xxxxx/xxx.pdf

Publication History

Doc. #	Date	Main Changes	Contributors
CAML Spec 1.0	April 2, 2001	Created Document	Ed Costello Dmitry Kirsanov, Thom Linden, Serge Swerdlow
CAML Spec 1.1	June 22, 2001	Added information on the segment elements	Alex Richter
CAML Spec 1.1	July 23, 2001	Changed according to transmission requirements	Dmitry Kirsanov
CAML Spec 1.1	July 31, 2001	Updated text	Thom Linden
CAML Spec 1.1	August 1, 2001	DTD updated	Dmitry Kirsanov

Table of Contents

CAML: Command Audio Markup Language Specification 1
Publication History 2
Table of Contents 3
1. CAML Language 4
1.1. Abstract 4
1.2. Introduction 5
2. Design Rationale 7
2.1. Navigation 7
2.2. Layout 7
2.3. User Interface 7
2.4. Timing and Synchronization 7
2.5. Media Object 8
2.6. Content Control 8
2.7. Accessibility 8
3. Definition of the CAML Language 8
3.1. Conformance Criteria 8
3.2. CAML Language 9
3.3. CAML Data References 10
3.3.1. CAML Data Reference Attributes 10
3.3.2. Layers and blocks 11
3.4. Navigation Module 11
3.4.1. The caml element 12
3.4.2. The content-instance element 12
3.4.3. The segment element 12
3.4.3.1. The segment element of EPG 13
3.4.3.2. The segment element of System Prompt program 14
4. A CAML document example 14
5. CAML Language Document Type Definition (DTD) 19
Appendix A 23
14. SMIL 2.0 Basic Language Profile 23
14.1. Abstract 23
14.2. Introduction 23
14.3. Design Rationale 24
14.3.1. Layout 24
14.3.2. User Interface 24
14.3.3. Timing and Synchronization 24
14.3.4. Media Object 25
14.3.5. Content Control 25
14.3.6. Accessibility 25
14.3.7. Use of the SMIL Basic language profile 25
14.4. Definition of the SMIL 2.0 Basic Language Profile 26
14.4.1. Conformance Criteria 26
14.4.2. SMIL 2.0 Basic Language Profile 27
14.4.3. Layout Modules 28
14.4.4. Linking Modules 28
14.4.5. Media Object Modules 29
14.4.6. Structure Module 29
14.4.7. Timing and Synchronization Modules 30
14.4.8. Content Control Modules 30
14.4.9. XML Namespace Declarations 31
14.5. SMIL 2.0 Basic Language Profile Document Type Definition (DTD) 31
References 36

1. CAML Language

1.1. Abstract

The Command Audio Markup Language (CAML) is designed to meet the Metadata definition needs of audio-centric interactive multimedia distributed over wireless digital audio broadcast (DAB) systems to mobile and stationary consumer devices. By Metadata, we generally mean data about content, such as the broadcast time of an audio program or that this edition of Nightline ™ has five stories. Using CAML, an author of a CAML document can combine and interrelate multiple audio-centric media objects into compelling presentations. In automobile driver and other 'eyes on the road, hands on the wheel' environments, only the audio layers of the rich media are expected to be available to consumers. Thus, the audio-centricity of the specification (as opposed to video-centric definitions such as MPEG4 or MPEG7) is a core fundamental requirement. The media objects may include

interactive capabilities, but these should not mandate a driver's attention (e.g. to play a media selection or to complete an advertised purchase). In order to experience these multimedia objects independent of the broadcast schedule (i.e. as on-demand or time shifted media), there is no requirement for real-time presentation.

DAB systems are distinguished in that they are one-way, point to multi-point transmission systems. These systems have an inherent, fundamental advantage over point to point transmission systems (e.g. wireless internet systems) by allowing a single broadcast to reach an unlimited number of receivers at no incremental bandwidth cost. DAB systems also exhibit unique data loss characteristics (e.g., data loss when obstacles block reception).

The CAML language is designed to meet the following criteria:

1. The language definition is open, flexible, extensible and able to represent complex object relationships.
2. A rich set of presentation capabilities is available to media authors.
3. The authored media is suitable for distribution over wireless DAB systems to mobile devices.
4. The presentation of CAML media is efficient and interoperable on low-power, portable consumer devices.
5. To obtain optimal advantage of DAB systems, CAML object distribution needs to be resilient to (some degree of) data loss.
6. It is suitable to define both program contents and electronic program guides (the penultimate metadata for broadcast audio -or video- services)

To achieve these criteria, the authors observed a similarity between the criteria for CAML and to Internet streaming media technologies such as RealServer and Real Player by RealNetworks, Inc. and Windows Media Server and Player by Microsoft, Inc. For streaming audio, these Internet technologies are audio-centric. They may incorporate other media as non-essential presentation elements. A noted difference is that Internet streaming media allows a server to choose alternative compression streams based on the dynamic performance of the Internet connection to each client. With DAB systems, there is no variance of bandwidth (performance) on a per client basis. An additional distinction is the desire to enable consumer navigation capability over the audio-centric object. That is, consumers should be able to follow audio-centric document relationships using simple navigational interfaces which do not require visual clues such as underlined or color-based textual linkages requiring a mouse or keyboard centric interface.

As a starting point, the authors chose the Synchronized Multimedia Integration Language (SMIL) 2.0 specification. Thus, CAML contains a subset of the SMIL 2.0 features including basic layout, linking, media object, structure, and timing modules, and a set of CAML unique extensions that enable rich navigation and semantic linking between media segments.

1.2. Introduction
This section is informative.

The design of wireless digital audio broadcast systems has traditionally focused on delivering high-fidelity real-time audio to consumers in mobile and stationary environments. Examples of such systems are:

? the satellite digital audio radio system (SDARS) developed by XM Satellite Radio
? the SDARS developed by Sirius Satellite Radio
? the terrestrial in-band, on-channel system developed by iBiquity Digital Corporation
? the European Telecommunications Standards Institute (ETSI) standard for Radio Broadcasting Systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers. EN 300 401 V1.3.2 (commonly known as Bureka-147)

The Command Audio Markup Language was defined to provide a uniform standard-based definition which extends these systems to allow services that deliver additional, richer, but still audio-centric, media for time-shifted presentation. Such services are termed On-Demand Interactive Audio (ODIA) services. The CAML language serves as a standard representation for multimedia On-Demand Interactive presentations over DAB systems. Note that systems which implement ODIA services can include capabilities to represent both real-time and non-real-time content and to allow time-shifted presentation of both.

CAML content providers may distribute their content over a variety of channels, the digital audio broadcast channels mentioned above, conventional AM or FM analog channels, or two-way wireless or wired channels. EPG providers may distribute their content over the same or different channels as the content providers.

CAML content providers may present their programs on a wide variety of clients, such as ODIA-enabled digital audio broadcast radio receivers, DAB receivers integrated with mobile phones, car navigation systems, and voice user agents. Each of these platforms (or integration of platforms) has its specific capabilities and may require its own flavor of CAML presentation. CAML does not define the actual device presentation, it defines the media object and relationships.

To achieve its interoperability and portability goals, CAML includes a number of capabilities from the Synchronized Multimedia Integration Language (SMIL) 2.0 defined at http://www.w3.org/TR/2001/WD-smil20-20010301/. The SMIL modularization provides a way to create profiles (subsets) of the full SMIL language, in addition to providing the means to integrate SMIL functionality into other languages. In particular, the set of SMIL modules used in CAML represent the SMIL Basic Profile as defined at http://www.w3.org/TR/2001/WD-smil20-20010301/smil-basic.html. But, SMIL 2.0 defines a number of capabilities (e.g. complex timing with video objects) which are not appropriate given the CAML objectives. Future editions of this document will identify specific basic profile elements are not required by CAML. In addition to the SMIL modules, CAML also includes extensions that enable rich navigation and semantic linking between audio-centric media segments.

This document presents a fixed CAML definition to maximize portability. We can anticipate that in the future, like SMIL profiles, a CAML profile for ODIA-enabled devices can be tailored based on the full CAML language with or without extensions to support application specific features.

CAML documents represent either the metadata associated with media to be listened-to by consumers (e.g. editions of audio programs) or the metadata describing the selections of media available to consumers. The metadata describing selections is generally referred to as an Electronic Program Guide (or EPG). Since the media experience of consumers of CAML described content is audio-centric, it is a complementary requirement that the EPG also be audio-centric. That is, the program guide used for personalization of a ODIA-enabled device should have a presentation which is consistent with all other CAML objects. In this respect, there are similarities between an EPG for audio-centric services and an EPG for video-centric services. The TV-Anytime Forum is creating international consensus on definitions for systems which enable video-on-demand capabilities. These include META definitions for a Program Guide. Where practical, we have chosen to adopt several of these features as part of the CAML definition.

It should be noted that the CAML definitions, SMIL definitions and TV-Anytime definitions, per se, make no statement regarding IPR regarding implementation of systems incorporating these features. It is anticipated that this document (or future versions thereof) will be contributed to standardization bodies such as WorldDAB (http://www.worlddab.org/), the Wireless Multimedia Forum (http://www.wmmforum.com/), and TV-Anytime Forum (http://www.tv-anytime.org/. Such submission makes no representation as to specific Command Audio or other organization's product capabilities, copyrights or intellectual property rights.

For information and specifications regarding TV-Anywhere the reader is referred to http://www.tv-anytime.org/.
For information and specifications regarding SMIL the reader is referred to http://www.w3.org/TR/REC-smil/2000/SMIL20/.

That a player is CAML conformant means that it can play documents at least as complex as those allowed by the CAML language definition.

This document is expected to expand in future version to incorporate meta information describing consumer metadata such as usage history, user preferences, and transactional media layers for interacting with 2-way backchannels.

2. Design Rationale

This section is informative.

CAML language may be supported by a wide variety of ODIA-enabled players running on multiple hardware platforms. Mobile and portable devices share some common characteristics:

LCD display: display can render only texts in a small area (for example a radio with two lines of twelve characters each).

Simple input method: Input devices may be functional, arrow keys, and a select key. Another may have a voice interpreter.

Audio-centric: The primary interaction with players is audio only. That is, no textual or graphical viewing is required to interact with the object rendering.

The CAML language aims to meet these requirements.

2.1. Navigation
A CAML document consists of one or more content items, represented in CAML language constructions. CAML constructions are used primarily to implement navigational capabilities of ODIA content. At the level of primary segments SMIL modules are used. Each SMIL module is an XML subtree which, taken separately, fully conforms to the definition of a SMIL 2.0 Basic Profile document.
2.2. Layout
Layout coordinates presentation of objects on the display device. Layout directives in CAML will be defined using SMIL modules (see Appendix A) The SMIL BasicLayout Module is used and the more complex functionality of the other layout modules, such as hierarchical layout regions, is not supported.
2.3. User Interface
In an ODIA player device, a user would likely use radio-like dials to select the target that activates playback or linking. A "mouse-like" pointing cursor device is generally not supported. While a user is handling the focus, the player may slow or pause its timeline; skip to the next segment; traverse to additional information and perform other navigational movements.

2.4. Timing and Synchronization
Timing and Synchronization aspects of CAML presentation are handled using SMIL modules (Appendix A). The SMIL Timing and Synchronization Modules present dynamic and interactive multimedia according to a timeline. The SMIL timing model is expressed in a structured language. The timeline of a SMIL Basic Profile presentation may need to be processed with limited memory and processing resources of mobile devices. For example, recursive function calls caused by nesting elements and memory allocation for additional timelines should be restricted. To achieve this, the CAML language has restrictions on use of the SMIL Timing.

The restrictions are:
Begin or end conditions are not allowed.

Non-root time containers are not supported.

Time attributes support the basic timing for an element. Timing attribute values support a synchronization-based timeline. Simple event timing is useful and is usually easy to support, and so is included in CAML.
2.5. Media Object
Location of the media that constitute the contents of the presentation in CAML is defined using SMIL module BasicMedia. The ODIA multimedia presentation is composed of media such as audio, text, images, animations and video.

2.6. Content Control
The CAML language includes the SMIL BasicContentControl Module. For the sake of authoring consistency, a CAML document should be able to contain several presentations for different kinds of clients, controlled by switch elements and test attributes.

The player should analyze switch element syntax correctly even if it cannot recognize the test attributes' values.

2.7. Accessibility

The document, "Accessibility Features of SMIL" [WAI-SMIL-ACCESS], summarizes their recommendations on the SMIL 1.0 [SMIL1].

3. Definition of the CAML Language
This section is normative.
3.1. Conformance Criteria
A CAML document is a "Conforming CAML Document" if it adheres to the specification described in this document including CAML DTD and also:

? The root element of the document is a caml element.

? Conforms to the "Extensible Markup Language (XML) 1.0" [XML10] and "Namespaces in XML" specifications, the document is well-formed.

? A document must declare a default namespace for its elements with its xmlns attribute at the caml root element. A CAML document is identified with http://www.tobedetermined URI.

For example:
<caml xmlns="http://www.tobedetermined">
...
</caml>

If you wish to use SMIL modules that are not specifically included in the CAML language definition, you must identify them as being from the SMIL 2.0 namespace. Certain limits on CAML extensions should be applied (to be defined)

A document may also identify itself as a valid CAML XML document with a DOCTYPE declaration, although a CAML document must still include the above CAML namespace identifier.

The CAML language DOCTYPE is:
<!DOCTYPE caml
PUBLIC "-//W3C//DTD SMIL 2.0 Basic//EN"
http://www.tobedetermined/CAML.dtd >
This DOCTYPE declares a valid, extension-free, CAML document.

The rules above will be updated once an XML Schema for CAML language is available.

Conforming CAML Players

The CAML player is a program that can parse and process a CAML document and render the contents of the document onto an output medium.

The following criteria apply to "Conforming CAML Players":

The player must conform to functionality criteria defined in each module in ways consistent with this profile specification.

The player should support the semantics of all CAML language features. The player ignores unknown elements under support of skip-content attributes. The player ignores unknown attributes.

For an attribute with an unknown attribute value, the player substitutes the default attribute value if any, or ignores the attribute if the attribute has no default value.

3.2. CAML Language

The CAML language supports the lightweight multimedia features defined in the SMIL 2.0 specification. It includes the following SMIL 2.0 Modules:

SMIL 2.0 Layout Modules -- BasicLayout
SMIL 2.0 Linking Modules -- BasicLinking
SMIL 2.0 Media Object Modules -- BasicMedia and MediaClipping
SMIL 2.0 Structure Module -- Structure
SMIL 2.0 Timing and Synchronization Modules -- BasicInlineTiming, SyncbaseTiming, EventTiming, MinMaxTiming, and BasicTimeContainers
SMIL 2.0 Content Control Modules -- BasicContentControl and SkipContentControl*

These collections of elements are used in the following sections defining the CAML content model:

Collection Name	Elements in Collection
LinkAnchor	a, area(anchor)
MediaContent	Text, img, audio, video, ref, animation, textstream
Schedule	Par, seq
EMPTY	

Collections of attributes used in the tables below are defined as follows:

Collection Name	Attributes in Collection
Core	Id, class, title
Timing	Begin, dur, end, repeatCount, repeatDur, min, Max
Description	Abstract, author, copyright
I18n	Xml:lang
Test	SystemBitrate(system-bitrate), systemCaptions(system- captions),systemLanguage(system-language), SystemOverdubOrSubtitle(system-overdub-or-caption), SystemRequired(system-required), systemScreenSize(system-screen-size), SystemScreenDepth(system-screen-depth), systemAudioDesc, SystemOperatingSystem, systemCPU, systemComponent

3.3. CAML Data References

CAML data references are special constructs that enable the CAML document to refer to specific parts of a chunk of binary data by specifying the starting offset and size of the part being referred. The goal of introducing this construct is to enable a CAML player to quickly locate binary data for media playback without having to scan large extents of binary data in search for specific boundary markers. It is recommended, therefore, that all binary data pertaining to a single CAML document be combined into one binary chunk that accompanies this document (although CAML allows to refer to more than one binary chunk using the chunks' IDs).

The specific mechanism of transferring binary data and associating binary chunks with CAML documents is implementation dependent; however, any conformant CAML player must be able to parse CAML data references and be able to locate the specified fragment of data in the binary chunk(s) which accompany the current CAML document.

3.3.1. CAML Data Reference Attributes

A CAML data reference consists of a number of attributes as defined below:

src
The name of the binary chunk. The value of src is normally equal to "caml:this", thus referring to the data chunk attached to the current CAML document. Other values of this attribute with "caml:" prefix refer to other binary chunks accessible to the document. Also, this attribute can contain any other valid URI which is processed according to the SMIL specification [SMIL2].

An example of a media object element with a src attribute using CAML data reference:

```
<audio id="9" src="caml:this" caml:offset="1067914" caml:length="53360" caml:channels="1" caml:samplerate="8000"
caml:bitspersample="16" caml:codectype="PCM" />
```

3.3.2. Layers and blocks

Audio layers consist of blocks which differ by the audio compression method used. Audio blocks are represented by <audio> elements and grouped together using<seq> elements corresponding to audio layers.

Along with the attributes described in the SMIL specification, the audio elements may have the following CAML-specific attributes:

id
A unique identifier of the block in the CAML document

caml:offset
The starting offset of the block, in bytes, relative to the start of the CAML binary chunk.

caml:length
The length of the block in bytes.

caml:playtime
The length of the block in milliseconds

caml:channels
The number of audio channels in the block, e.g. 1 for mono, 2 for stereo audio.

caml:samplerate
The number of samples per second in the audio block.

caml:bitspersample
The number of bits per sample in the audio block.

caml:codectype
The compression type of the audio block. Possible values are:

The example of an audio layer with block information:

```
<seq title="H1-1.wav">
```

```
  <audio id="9" src="caml:this" caml:offset="1067914" caml:length="53360" caml:channels="1" caml:samplerate="44100"
caml:bitspersample="16" caml:codectype="DVSI" />
   <audio id="10" src="caml:this" caml:offset="1121274" caml:length="53360" caml:channels="1" caml:samplerate="8000"
caml:bitspersample="16" caml:codectype="Dolby" />
  </seq>
```

3.4. Navigation Module
The Navigation module is the top layer of a CAML document providing means for navigating across media segments (SMIL modules).

Elements	Attributes	Content Model
Caml	Core, I18n, caml-length, dialect	(content-item+)
Content-item	Core, I18n, type, contentid, expiration-time, edition-time, announce-time	(segment+)
Segment	Core, I18n, next_story, prev_story, next_primesegment, prev_primesegment, down_level, up_level, first, last, level, assoc_content_id, cif_type, EPG_title, status_mask, availability_prompt, prompt_alias	Smil

3.4.1. The caml element
The caml element is the root element of a CAML document. It contains content-item elements and has the following attributes:

caml-length
Length of the CAML document in bytes.

dialect
The identifier of a CAML dialect used in this content item, e.g. "Eureka147". (reserved for future use)
3.4.2. The content-instance element
The content-item element represents any version of a program (as per TV-Anytime) that has been created by applying minor editorial changes to the content. (e.g. editing out explicit material) It contains at least one segment element.

type
Content type, e.g. "program", "EPG" or "system_prompts".

contentid
Unique ID of the content item identifying the program in the content database.

expiration-time
Time when the content is considered expired, in the format "MM/DD/YYYY HH:MN:SS".

edition-time
Time when the content item was last edited, in the format "MM/DD/YYYY HH:MM:SS".

announce-time
Time when the content item is to be available for transmission, in the format "MM/DD/YYYY HH:MM:SS".
3.4.3. The segment element
The attributes of the segment element are defined as follows:

down_level
The value of the id attribute of a segment element one level down from the current segment in a multi-level program.

up_level
The value of the id attribute of a segment element one level up from the current segment in a multi-level program. This attribute might be redundant (tbd).

prev_story
The value of the id attribute of the first segment element in the previous story of the program.

next_story
The value of the id attribute of the first segment element in the next story of the program.

prev_primesegment
The value of the id attribute of the previous segment element in the current story, or, if it is the first segment of the story, it is equivalent to prev_story.

next_primesegment
The value of the id attribute of the next segment element in the current story, or, if it is the last segment of the story, it is equivalent to next_story.

first
The value of this attribute is equal to "yes" if it is the first segment of a story.

last
The value of this attribute is equal to "yes" if it is the last segment of a story.

level
The value of this attribute is equal to 0 for segment elements belonging to the first level of a multi-level program, and it is equal to 1 for segment elements belonging to the second level.

3.4.3.1. The segment element of EPG
The segment element of a content item of type 'EPG' has additional attributes:

assoc_content_id
ContentId of a topic/program associated with the content of the segment.

cif_type
Content class type, which can have the following values:
? Program
? EPG
? System_prompt
? Traffic_bulletin
? News_bulletin
? Emergency bulletin
? Undefined

status_mask
A flag defining specialized rendering of the EPG item associated with the segment. Possible values are:
? disable_save
? invisible_on_epg

availability_prompt

Reference to the system availability prompt (id of the segment containing the prompt in "system prompts" program)
3.4.3.2. The segment element of System Prompt program
The segment element of a content item of type "system_prompts" has two attributes in addition to attributes common to all content items:

prompt_alias
An alias name of the system prompt associated with the segment.

assoc_content_id
ContentId of a prompt associated with the content of the segment.

4. A CAML document example

The structure shown on this diagram is implemented in the following CAML document:

```
<?xml version="1.0" encoding="us-ascii" ?>
- <caml version="1.12" xmlns:caml="http://www.commandaudio.com/caml" dialect="navigator" caml-length="5444">
- <content-item type="program" title="TestCase5" contentid="99999" expiration-time="06/30/2003 00:00:00" edition-time="07/23/2001 23:43:22" announce-time="07/23/2001 23:43:22">
- <segment id="3" next_story="17" next_primesegment="17" down_level="7" level="0" last="yes" first="yes">
- <smil xmlns="http://www.w3.org/TR/REC-smil">
- <body>
- <par>
- <seq title="h1_1.txt">
  <text id="4" src="caml:this" caml:offset="0" caml:length="127" />
  </seq>
- <seq title="h1_1.wav">
  <audio id="6" src="caml:this" caml:offset="127" caml:length="1891020" caml:channels="2" caml:samplerate="44100" caml:bitspersample="16" caml:codectype="PCM" />
  </seq>
  </par>
  </body>
  </smil>
  </segment>
- <segment id="7" next_story="17" next_primesegment="11" level="1" first="yes">
- <smil xmlns="http://www.w3.org/TR/REC-smil">
- <body>
- <par>
- <seq title="d1_1.txt">
  <text id="8" src="caml:this" caml:offset="1891147" caml:length="179" />
  </seq>
- <seq title="d1_1.wav">
  <audio id="10" src="caml:this" caml:offset="1891326" caml:length="2281372" caml:channels="2" caml:samplerate="44100" caml:bitspersample="16" caml:codectype="PCM" />
  </seq>
  </par>
  </body>
  </smil>
  </segment>
- <segment id="11" next_story="17" next_primesegment="17" prev_primesegment="7" level="1" last="yes">
- <smil xmlns="http://www.w3.org/TR/REC-smil">
- <body>
- <par>
- <seq title="d1_2.txt">
  <text id="12" src="caml:this" caml:offset="4172698" caml:length="107" />
  </seq>
- <seq title="d1_2_S.wav">
  <audio id="14" src="caml:this" caml:offset="4172805" caml:length="429248" caml:channels="2" caml:samplerate="44100" caml:bitspersample="16" caml:codectype="PCM" />
  </seq>
  </par>
  </body>
  </smil>
  </segment>
- <segment id="17" next_story="31" next_primesegment="21" prev_primesegment="3" prev_story="3" down_level="25" level="0" first="yes">
- <smil xmlns="http://www.w3.org/TR/REC-smil">
- <body>
- <par>
- <seq title="h2_1.txt">
  <text id="18" src="caml:this" caml:offset="4602053" caml:length="76" />
  </seq>
- <seq title="h2_1.wav">
  <audio id="20" src="caml:this" caml:offset="4602129" caml:length="307388" caml:channels="2" caml:samplerate="44100" caml:bitspersample="16" caml:codectype="PCM" />
  </seq>
  </par>
  </body>
  </smil>
  </segment>
- <segment id="21" next_story="31" next_primesegment="31" prev_primesegment="17" prev_story="3" down_level="25" level="0" last="yes">
- <smil xmlns="http://www.w3.org/TR/REC-smil">
- <body>
- <par>
- <seq title="h2_2.txt">
  <text id="22" src="caml:this" caml:offset="4909517" caml:length="117" />
  </seq>
- <seq title="h2_2.wav">
  <audio id="24" src="caml:this" caml:offset="4909634" caml:length="1493416" caml:channels="2" caml:samplerate="44100" caml:bitspersample="16" caml:codectype="PCM" />
  </seq>
  </par>
  </body>
  </smil>
  </segment>
- <segment id="25" next_story="31" next_primesegment="31" prev_primesegment="3" prev_story="3" level="1" last="yes" first="yes">
- <smil xmlns="http://www.w3.org/TR/REC-smil">
- <body>
```

```
- <par>
- <seq title="d2_1.txt">
  <text id="26" src="caml:this" caml:offset="6403050" caml:length="88" />
  </seq>
- <seq title="d2_1.wav">
  <audio id="28" src="caml:this" caml:offset="6403138" caml:length="4386480" caml:channels="2" caml:samplerate="44100"
caml:bitspersample="16" caml:codectype="PCM" />
  </seq>
  </par>
  </body>
  </smil>
  </segment>
- <segment id="31" next_primesegment="35" prev_primesegment="17" prev_story="17" down_level="39" level="0" first="yes">
- <smil xmlns="http://www.w3.org/TR/REC-smil">
- <body>
- <par>
- <seq title="h3_1.txt">
  <text id="32" src="caml:this" caml:offset="10789618" caml:length="105" />
  </seq>
- <seq title="h3_1.wav">
  <audio id="34" src="caml:this" caml:offset="10789723" caml:length="490536" caml:channels="2" caml:samplerate="44100"
caml:bitspersample="16" caml:codectype="PCM" />
  </seq>
  </par>
  </body>
  </smil>
  </segment>
- <segment id="35" prev_primesegment="31" prev_story="17" down_level="39" level="0" last="yes">
- <smil xmlns="http://www.w3.org/TR/REC-smil">
- <body>
- <par>
- <seq title="h3_2.txt">
  <text id="36" src="caml:this" caml:offset="11280259" caml:length="147" />
  </seq>
- <seq title="h3_2.wav">
  <audio id="38" src="caml:this" caml:offset="11280406" caml:length="355572" caml:channels="2" caml:samplerate="44100"
caml:bitspersample="16" caml:codectype="PCM" />
  </seq>
  </par>
  </body>
  </smil>
  </segment>
- <segment id="39" next_primesegment="43" prev_primesegment="17" prev_story="17" level="1" first="yes">
- <smil xmlns="http://www.w3.org/TR/REC-smil">
- <body>
- <par>
- <seq title="d3_1.txt">
  <text id="40" src="caml:this" caml:offset="11635978" caml:length="53" />
  </seq>
- <seq title="d3_1.wav">
  <audio id="42" src="caml:this" caml:offset="11636031" caml:length="2803056" caml:channels="2" caml:samplerate="44100"
caml:bitspersample="16" caml:codectype="PCM" />
  </seq>
  </par>
  </body>
  </smil>
  </segment>
- <segment id="43" prev_primesegment="39" prev_story="17" level="1" last="yes">
- <smil xmlns="http://www.w3.org/TR/REC-smil">
- <body>
- <par>
- <seq title="d3_2.txt">
  <text id="44" src="caml:this" caml:offset="14439087" caml:length="229" />
  </seq>
- <seq title="d3_2.wav">
  <audio id="46" src="caml:this" caml:offset="14439316" caml:length="562692" caml:channels="2" caml:samplerate="44100"
caml:bitspersample="16" caml:codectype="PCM" />
  </seq>
  </par>
  </body>
  </smil>
  </segment>
  </content-item>
  </caml>
```

5. CAML Language Document Type Definition (DTD)
This section is normative.

```
<!-- ============================================================== -->
<!-- CAML DTD ====================================================== -->
<!-- file: caml.dtd

This is Command Audio Markup Language (CAML)

Copyright 2001 Command Audio Corporation. All Rights Reserved.

Permission to use, copy, modify and distribute the CAML DTD and its accompanying documentation for any purpose and without fee is
hereby granted in perpetuity, provided that the above copyright notice and this paragraph appear in all copies.  The copyright
holders make no representation about the suitability of the DTD for any purpose. It is provided "as is" without expressed or
implied warranty.

Author:     CAC
Revision:   $Id: caml.dtd,v 0.1 2000/11/10 11:16:46  Exp $

Please use this formal public identifier to identify this DTD:

        "-//CommandAudio//DTD CAML//EN"

-->
```

```
<!- First we include the necessary SMIL modules.
       CAML supports the lightweight multimedia
        features defined in SMIL language. This profile includes
        the following SMIL modules:

             SMIL 2.0 BasicLayout Module
             SMIL 2.0 BasicLinking Module
             SMIL 2.0 BasicMedia and MediaClipping Modules
             SMIL 2.0 Structure Module
             SMIL 2.0 BasicInlinTiming, SyncbaseTiming, SventTiming,
                  MinMaxTiming and BasicTimeContainers Modules
             SMIL 2.0 BasicContentControl and SkipContentControl Modules
-->

<!ENTITY % NS.prefixed "IGNORE" >
          <!ENTITY % SMIL.prefix "" >

          <!-- Define the Content Model -->
          <!ENTITY % smil-model.mod
              PUBLIC "-//W3C//ENTITIES SMIL 2.0 Document Model 1.0//EN"
                     "smil-model-1.mod" >

          <!-- Modular Framework Module  ............................... -->
          <!ENTITY % smil-framework.module "INCLUDE" >
          <![%smil-framework.module;[
          <!ENTITY % smil-framework.mod
               PUBLIC "-//W3C//ENTITIES SMIL 2.0 Modular Framework 1.0//EN"
                      "smil-framework-1.mod" >
          %smil-framework.mod;]]>

<!ENTITY % layout-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Layout//EN"
  "SMIL-layout.mod">
%layout-mod;

<!ENTITY % link-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Linking//EN"
  "SMIL-link.mod">
%link-mod;
<!ENTITY % BasicLinkingModule "INCLUDE">

<!ENTITY % media-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Media Objects//EN"
  "SMIL-media.mod">
%media-mod;

<!ENTITY % struct-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Document Structure//EN"
  "SMIL-struct.mod">
%struct-mod;

<!ENTITY % timing-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Timing//EN"
  "SMIL-timing.mod">
%timing-mod;

<!ENTITY % control-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Content//EN"
  "SMIL-control.mod">
%control-mod;


<!-Now we define the CAML elements-->

<!ELEMENT caml  (content-item+)>
<!ATTLIST caml   %Core.attrib;
                 %I18n.attrib;
                 caml-length          CDATA     #REQUIRED
                 dialect              CDATA     #REQUIRED

Caml      Core, I18n, caml-length, dialect
Content-item       Core, I18n, type, contentid, expiration-time, edition-time, announce-time

Segment   Core, I18n, next_story, prev_story, next_primesegment, prev_primesegment, down_level, up_level, first, last, level,
assoc_content_id, cif_type, EPG_title, status_mask, availability_prompt, prompt_alias



<!ELEMENT content-item  (segment+)>
<!ATTLIST content-item
  %Core.attrib;
              %I18n.attrib;
              type                 CDATA     #REQUIRED
              contentid            CDATA     #REQUIRED
              prime-segment-count  CDATA     #REQUIRED
              expiration-time      CDATA     #REQUIRED
              edition-time         CDATA     #REQUIRED
              announce-time        CDATA     #REQUIRED


<!ELEMENT segment  (smil)>
<!ATTLIST segment %Core.attrib;
                  %I18n.attrib;
                        up_level        IDREF  #IMPLIED
                        down_level      IDREF  #IMPLIED
                        prev_story      IDREF  #IMPLIED
                        next_story      IDREF  #IMPLIED
                        next_primesegment   IDREF  #IMPLIED
                        prev_primesegment IDREF  #IMPLIED
```

```
        first          'yes'  #IMPLIED
last            'yes'  #IMPLIED
        'level          CDATA #IMPLIED
    assoc_content_id IDREF #IMPLIED
cif_type        CDATA #IMPLIED
EPG_title       CDATA #IMPLIED
status_mask     CDATA #IMPLIED
availability_prompt    CDATA #IMPLIED
prompt_alias        CDATA #IMPLIED
>

<!-end of CAML DTD ==========================================-->
```

An XML Schema for CAML will be made available.

Appendix A

14. SMIL 2.0 Basic Language Profile

Copyright 2000 World Wide Web Consortium (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University. All Rights Reserved. http://www.w3.org/TR/smil20/smil-basic.html

Editors
Kenichi Kubota (kuboken@isl.mei.co.jp), Panasonic
Aaron Cohen (aaron.m.cohen@intel.com), Intel

14.1. Abstract

The SMIL 2.0 Basic language profile defines a profile that is a baseline subset of the SMIL 2.0 language profile specification, and is designed to be a profile that meets the needs of resource-constrained devices such as mobile phones and portable disc players. It contains a baseline set of the SMIL 2.0 features including basic layout, linking, media object, structure, and timing modules.

14.2. Introduction

This section is informative.

The Synchronized Multimedia Integration Language (SMIL, pronounced "smile") includes powerful functionality for multimedia services not only on desktops but also for information appliances. The SMIL 2.0 Basic language profile should serve such a ubiquitous network as a baseline for audio-visual presentations.

SMIL content authors may desire their works to be available on a widespread variety of web clients, such as desktops, television sets, mobile phones, portable disc players, car navigation systems, and voice user agents. Each of these platforms has its specific capabilities and may require its own profile. The SMIL modularization provide a solution to create profiles and extensions of the full SMIL language profile, in addition to providing the means to integrate SMIL functionality into other languages.

The SMIL Basic language profile consists of a small number of modules chosen from the complete set of SMIL 2.0 modules to provide a baseline in terms of semantics and syntax, and assures conformance to the larger SMIL language profile specification. A profile for mobile and portable devices can be tailored based on the SMIL Basic language profile with or without extensions to support application specific features.

The SMIL Basic language profile does not propose to restrict extension, but aims at a baseline of conformance between the full SMIL language profile and one appropriate for mobile and portable services. Saying that a player is SMIL Basic conformant, means that it can play documents at least as complex as those allowed by the SMIL Basic language profile. It may play significantly more complex documents. In particular, the browsers conforming to the SMIL 2.0 language profile will be automatically SMIL Basic compliant. Thus SMIL authoring for low power devices is easier since general tools can be used.

14.3. Design Rationale

This section is informative.

SMIL Basic language profile is a language profile that is SMIL Host Language Conformant and may be supported by wide variety of SMIL players, even those running on small mobile phones. Mobile and portable devices share some common characteristics:

? Small display: Display can render texts, images, audio and stream data in a small area.

? Simple input method: Input devices may be numeric keys, arrow keys, and a select key. Some may have a pointing cursor. Another may have a voice interpreter.

? Real-time embedded OS: Resources for calculation is limited by priority order of each task. So, in a SMIL player, the use of timers should be restricted in number and frequency, and memory should be used sparingly.

? Less network transaction: Network transactions, if necessary in a wireless environment, should be reduced as much as possible.

The SMIL Basic language profile aims to meet these requirements.

For references, see the document, "HTML4.0 Guidelines for Mobile Access" [MOBILE-GUIDE] issues guidelines for HTML content, and "XHTML Basic" [XHTML-BASIC] provides a minimum subset for portability and conformance.

14.3.1. Layout

Layout coordinates presentation of objects on the display device. Presentation on a small display has difficulty in rendering some objects. Layout of objects may not be able to be flexibly adjusted, nor have scroll bars. So, the layout should be simple and effective. Often the root-layout window will represent the full screen of the display. The BasicLayout Module is used and the more complex functionality of the other layout modules, such as hierarchical layout regions, is not supported.

14.3.2. User Interface

On a SMIL Basic player window, a user would likely use arrow keys to move focus on objects and anchors, and select the target that activates playback or linking. A "mouse-like" pointing cursor device might not be supported. A "move focus and select" is a simple user interface for communication with the SMIL player. The "mouse-click" user action can be mapped to an "activate" action. While a user is handling the focus, the player may slow or pause its timeline.

14.3.3. Timing and Synchronization

The SMIL Timing and Synchronization Modules present dynamic and interactive multimedia according to a timeline. The SMIL timing model is expressed in a structured language. The timeline of SMIL Basic presentation may need to be processed with limited memory and processing resources of mobile devices. For example, recursive function calls caused by nesting elements and memory allocation for additional timelines should be restricted.

To achieve this, the SMIL Basic language profile has restrictions on use of the SMIL Timing. The restrictions are:

Time attribute values only allow single begin and end conditions. Lists of begin or end conditions are not allowed.

@@ In addition, the SMIL Basic profile may need to have no concept of a time container except a root time container. A time container groups media objects into a basic unit within a local time space, and this may increase the processing complexity of the document beyond device capabilities. If the document includes time containers, the nesting depth of time containers may need to be limited. Also it may be required to limit the number of characters or elements. The SYMM Working Group requests feedback from users of SMIL Basic on these points.

Time attributes support the basic timing for an element. Timing attribute values support a synchronization-based timeline. Enriched players may support event values, for example, like "click" to start presentation. This kind of simple event timing is useful and is usually easy to support, and so is included in SMIL Basic profile.

14.3.4. Media Object

The BasicMedia Module specifies the location of the media that constitute the contents of the presentation. Here we assume the presentation to be an audio/visual multimedia presentation composed of media such as text, images, audio and video.

14.3.5. Content Control
The SMIL Basic profile includes the BasicContentControl Module. For the sake of authoring consistency, a SMIL document should be able to contain several presentation for different kinds of clients, controlled by switch elements and test attributes. A client application may not need to directly support content control mechanisms, since the content control processing of the document may be done on the way to the client. When SMIL Basic is used in this way, the player may be thought of as the system composed of the combination of the server and the client.

Test attributes of the BasicContentControl Module are included in the SMIL Basic language profile, even when an alternative to the CC/PP mechanism is necessary. The player should analyze switch element syntax correctly even if it cannot recognize the test attributes.

Some functionality of the Content Control Modules may be extended with CC/PP [CC/PP] mechanisms, which provide a way to notify device capabilities and user preferences to an origin server and/or intermediaries such as a gateway or proxy. This allows the generation or selection of device-tailored contents. Thus CC/PP can be used with transformation of available documents between client and server.

14.3.6. Accessibility
There are useful guidelines to handling accessibility issues that can be applied to the profile design for information appliances. The document, "Accessibility Features of SMIL" [WAI-SMIL-ACCESS], summarizes their recommendations on the SMIL 1.0 [SMIL1]

14.3.7. Use of the SMIL Basic language profile
SMIL Basic language profile is designed to be appropriate for small information appliances like mobile phones and portable disc players. Its simple design is intended to serve as a baseline profile for application specific extensions. Similar to the XHTML+SMIL language profile, an HTML+SMIL mobile profile could be written that is an integration of SMIL Basic timing and media functionality into the HTML mobile profile.

The SMIL Basic language profile does not restrict extension and inclusion of other modules, for example, the Animation, Metainformation, Transition, or additional Timing and Synchronization Modules. A language may support additional features and modules and still be conformant with the SMIL Basic language profile.
14.4. Definition of the SMIL 2.0 Basic Language Profile
This section is normative.
14.4.1. Conformance Criteria
Conforming SMIL 2.0 Basic Documents

A SMIL 2.0 Basic document is a "Conforming SMIL 2.0 Basic Document" if it adheres to the specification described in this document including SMIL 2.0 Basic DTD and also:

The root element of the document is a smil element.
Conforms to the "Extensible Markup Language (XML) 1.0" [XML10] specification, the document is well-formed.

A document must declare a default namespace for its elements with its xmlns attribute at the smil root element. The SMIL 2.0 Basic language profile document is identified with "http://www.w3.org/TR/REC-smil/2000/SMIL20/Basic" URI. For example:

```
<smil xmlns="http://www.w3.org/TR/REC-smil/2000/SMIL20/Basic">
  ...
</smil>
```

To use modules that are not specifically included in the SMIL 2.0 Basic language profile, they must be identified as being from the SMIL 2.0 namespace.

For example, a SMIL 2.0 Basic document extended to use the brush element from the SMIL 2.0 BrushMedia Module:

```
<smil xmlns="http://www.w3.org/TR/REC-smil/2000/SMIL20/Basic"
      xmlns:smil2="http://www.w3.org/TR/REC-smil/2000/SMIL20/">
  ...
  <smil2:brush color="red" begin="10s" dur="20s"/>
</smil>
```

A document may additionally identify itself as a valid SMIL XML document with a SMIL DOCTYPE declaration, although a SMIL 2.0 Basic document must still include the above SMIL 2.0 Basic namespace identifier.

The SMIL 2.0 Basic language profile DOCTYPE is:

```
<!DOCTYPE SMIL
          PUBLIC "-//W3C//DTD SMIL 2.0 Basic//EN"
          "http://www.w3.org/TR/REC-smil/2000/SMIL20Basic.dtd">
```

This DOCTYPE declares a valid, extension-free, SMIL 2.0 Basic document. The rules above will be updated once an XML Schema for SMIL 2.0 Basic language profile is available.

Conforming SMIL 2.0 Basic Players

A SMIL 2.0 Basic player is a program that can parse and process a SMIL 2.0 Basic document and render the contents of the document onto an output medium.

The following criteria apply to a "Conforming SMIL 2.0 Basic Players":

? The player must parse a SMIL Basic document and evaluate its well-formedness conformant to the XML1.0 Recommendation [XML10].

? The player must conform to functionality criteria defined in each module in ways consistent with this profile specification.

? The player should support the semantics of all SMIL 2.0 Basic language profile features.

? The player ignores unknown elements under support of skip-content attributes.

? The player ignores unknown attributes.

? For an attribute with an unknown attribute value, the player substitutes the default attribute value if any, or ignores the attribute if the attribute has no default value.

14.4.2. SMIL 2.0 Basic Language Profile
The SMIL 2.0 Basic language profile supports the lightweight multimedia features defined in the SMIL 2.0 specification. This language profile includes the following SMIL 2.0 Modules:
? SMIL 2.0 Layout Modules -- BasicLayout*
? SMIL 2.0 Linking Modules -- BasicLinking*
? SMIL 2.0 Media Object Modules -- BasicMedia* and MediaClipping
? SMIL 2.0 Structure Module -- Structure*
? SMIL 2.0 Timing and Synchronization Modules -- BasicInlineTiming*, SyncbaseTiming*, EventTiming, MinMaxTiming*, and BasicTimeContainers*
? SMIL 2.0 Content Control Modules -- BasicContentControl* and SkipContentControl*
(*) = required modules in order to be SMIL Host Language Conformant.
These collections of elements are used in the following sections defining the SMIL 2.0 Basic profile's content model:

Collection Name	Elements in Collection
LinkAnchor	a, area(anchor)
MediaContent	text, img, audio, video, ref, animation, textstream
Schedule	par, seq
EMPTY	

Collections of attributes used in the tables below are defined as follows:

Collection Name	Attributes in Collection
Core	id, class, title
Timing	begin, dur, end, repeat(deprecated), repeatCount, repeatDur, min, max
Description	abstract, author, copyright
I18n	xml:lang
Test	systemBitrate(system-bitrate), systemCaptions(system-captions), systemLanguage(system-language), systemOverdubOrSubtitle(system-overdub-or-caption), systemRequired(system-required), systemScreenSize(system-screen-size), systemScreenDepth(system-screen-depth), systemAudioDesc, systemOperatingSystem, systemCPU, systemComponent

14.4.3. Layout Modules

The Layout Module provides a framework for spatial layout of visual components. The SMIL 2.0 Basic profile includes the SMIL 2.0 BasicLayout Module. This profile requires layout type of "text/smil-basic-layout".

Elements	Attributes	Content Model
layout	Core, I18n, type	(root-layout?, region*)
region	Core, I18n, backgroundColor(background-color), height, width, bottom, fit, left, right, top, z-index, showBackground	EMPTY
root-layout	Core, I18n, backgroundColor(background-color), height, width	EMPTY

Default of the size and position attributes is the full screen available to a client player. The default value of the fit attribute is "hidden". So when a player cannot layout a media object of target, then it presents the object as is inside the region.

14.4.4. Linking Modules

The Linking Module describes the hyperlink relationships between content. The SMIL 2.0 Basic profile includes the SMIL 2.0 BasicLinking Module.

A SMIL Basic player may not be able to control links between the source and the destination object, for example, in applying sourceLevel and destinationLevel attributes to volume. In this case it is permitted that the attributes of target control, which are listed in the table below, are just ignored. As a consequence, the attributes of the SMIL 2.0 BasicLinking Module might be replaced with their default attributes. The default value for show attribtue is "replace" and the sourcePlaystate attribute is ignored, see their definitions for detail.

The area element and the deprecated anchor element may not need to be supported if the device does not have an appropriate user interface, in which case the SMIL Basic player should play the presentation without the area map.

Elements	Attributes	Content Model
a	Core, I18n, href, show, sourcePlaystate, destinationPlaystate, sourceLevel, destinationLevel, accesskey, tabindex, target, external, actuate	MediaContent*
area(anchor)	Core, I18n, Timing, alt, coords, href, show, sourcePlaystate, destinationPlaystate, sourceLevel, destinationLevel, accesskey, tabindex, target, external	EMPTY

14.4.5. Media Object Modules

The Media Object Modules provide a framework for declaring media that constitute the contents of a SMIL presentation, and specify their locations. The SMIL 2.0 Basic profile includes the SMIL 2.0 BasicMedia and MediaClipping Modules.

Elements	Attributes	Content Model
text, img, audio, video, animation, textstream, ref	Core, I18n, Description, Timing, Test, src, region, fill, alt, longdesc, type, clipBegin(clip-begin), clipEnd(clip-end)	area(anchor)

Elements of the BasicMedia Module have attributes describing basic timing properties of contents. For timing, the begin and end attributes should have one attribute value for a single timeline.

14.4.6. Structure Module

The Structure Module describes the structure of the SMIL document. The SMIL 2.0 Basic profile includes the SMIL 2.0 Structure Module.

The structure element body is implicitly defined to be a seq time container as in the SMIL1.0 and SMIL 2.0 language profiles, and this is true in SMIL 2.0 Basic profile as well.

Elements	Attributes	Content Model
smil	Core, I18n	(head?, body?)
head	Core, I18n	(layout? | switch?)
body	Core, I18n	(Schedule | switch | MediaContent | LinkAnchor)*

14.4.7. Timing and Synchronization Modules

The Timing and Synchronization Modules provide a framework for describing timing structure, timing control properties, and temporal relationships between elements. The SMIL 2.0 Basic profile includes the basic functionality of the SMIL 2.0 Timing and Synchronization Modules (SMIL Timing Modules). It is based upon the SMIL 2.0 BasicInlineTiming, SyncbaseTiming, EventTiming, and BasicTimeContainers modules.

@@The SYMM Working group desires feedback on the need to limit the complexity of SMIL Basic documents in order to ensure that they can be played on low power devices. Until recently, we have been considering limiting the use of time containers to one top-level container, but this now appears both insufficient in limiting document complexity and too restrictive to authors. We would like feedback from implementers on what limitations on complexity are necessary.

Time containers are basic units in synchronization defined in the SMIL Timing Modules and they group elements with synchronization relationships. In a SMIL Basic document with single time container, par and seq time container elements contain media object elements and should not have other time container elements nested inside them.

Attributes of the SMIL Timing Modules apply to elements of the SMIL 2.0 BasicMedia Module. Basics of timing are described in the SMIL Timing Modules. Control of element start/end time is available with the begin, dur, and end attributes. For repeating elements, the deprecated repeat attribute, the repeatCount and repeatDur attributes are available.

The begin and end attributes contain the offset-value, syncbase-value, smil-1.0-syncbase-value, and event-value attribute values as single conditions. The SMIL Basic profile includes the EventTiming Module, which includes "activateEvent" or "click". These events are mapped to "activate" user action to select a focused element in a SMIL Basic player.

As for the SMIL Timing Modules for the SMIL 2.0 Basic, there is only one attribute value in begin and end attributes, not a semicolon-separated list, in order to create a single flat timeline.

Elements	Attributes	Content Model
par	Core, I18n, Description, Test, Timing, endsync, fill, region	(switch | MediaContent | Schedule | LinkAnchor)*
seq	Core, I18n, Description, Test, Timing, fill, region	(switch | MediaContent | Schedule | LinkAnchor)*

The default value of the endsync attribute is "last". The default value of the fill attribute depends on the element type as described in the SMIL Timing Modules.

14.4.8. Content Control Modules

The Content Control Module provides a framework for selecting content. The SMIL 2.0 Basic profile includes the SMIL 2.0 BasicContentControl and SkipContentControl Modules. The skip-content attribute, which default value is "true", applies to SMIL 2.0 elements that were not a part of SMIL 1.0, and to elements that it was applied to in SMIL 1.0, and to elements currently empty in SMIL 2.0 but could be made non-empty in the future, following the definition in the module.

A SMIL Basic player must process the syntax of a switch element correctly even if the player could not evaluate the Test attribute in a media object element properly as the attribute implies. Test attributes that cannot be evaluated will default to false.

Elements	Attributes	Content Model
switch	Core, I18n	((LinkAnchor | MediaContent)* | Schedule* | layout*)

14.4.9. XML Namespace Declartions

XML Namespace declarations using the xmlns attribute are supported on all elements in the SMIL Basic language profile.
14.5. SMIL 2.0 Basic Language Profile Document Type Definition (DTD)
This section is normative.
The SMIL 2.0 Basic language profile DTD is defined as a set of SMIL 2.0 modules integrated.

```
<!-- ============================================================== -->
<!-- SMIL Basic Document Module Module============================== -->
<!-- file: smilbasic-model-1.mod

        This is SMIL 2.0 Basic, a proper subset of SMIL 2.0.
     Copyright 2000 W3C (MIT, INRIA, Keio), All Rights Reserved.

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

     PUBLIC "-//W3C//ENTITIES SMIL 2.0 Basic Document Model 1.0//EN"
     SYSTEM "smilbasic-model-1.mod"

     Author: Kenichi Kubota, Warner ten Kate, Jacco van Ossenbruggen
     Revision: $Id: smilbasic-model-1.mod,v 1.27 2000/09/21 03:10:02 kkubota Exp $
     ============================================================== -->

<!--
This file defines the SMIL 2.0 Basic Document Model.
All attributes and content models are defined in the second half of this file.  We first start with some utility definitions.
These are mainly used to simplify the use of modules in the second part of the file.

-->

<!-- ================== Util: Body - Media ============================ -->
<!ENTITY % media-object "text|img|audio|video|animation|textstream|ref">

<!-- ================== Util: Body - Content Control ================== -->
<!ENTITY % content-control "switch">

<!--================== Util: Body - Linking ============================ -->
<!ENTITY % BasicLink "a|anchor|area">
<!ENTITY % link "%BasicLink;">

<!-- ================================================================== -->
<!-- ================================================================== -->
<!-- ================================================================== -->

<!--
The actual content model and attribute definitions for each modulesection follow below.
-->

<!-- ================== Timing ======================================== -->
<!ENTITY % BasicInlineTiming.module    "INCLUDE">
<!ENTITY % MinMaxTiming.module         "INCLUDE">
<!ENTITY % SyncbaseTiming.module       "INCLUDE">
<!ENTITY % EventTiming.module          "INCLUDE">
<!ENTITY % BasicTimeContainers.module "INCLUDE">

<![%BasicInlineTiming.module;[
  <!ENTITY % basicTimeContainers "par|seq">
  <!ENTITY % timeContainer "%basicTimeContainers;">
  <!ENTITY % basicTimeContainers.content
   "(%media-object;|%content-control;|%link;|%basicTimeContainers;)*">
  <!ENTITY % timeContainer.content "%basicTimeContainers.content;">
  <!ENTITY % basicInlineTiming.attrib "%BasicInlineTiming.attrib;">
  <!ENTITY % basicTimeContainers.attrib "%BasicTimeContainers.attrib;">
]]>
<!ENTITY % timeContainer "">
<!ENTITY % timeContainer.content "">
<!ENTITY % basicInlineTiming "">
<!ENTITY % basicTimeContainers.attrib "">

<![%MinMaxTiming.module;[
  <!ENTITY % minMaxTiming.attrib "%MinMaxTiming.attrib;">
]]>
<!ENTITY % minMaxTiming.attrib "">

<!ENTITY % smil-time.attrib "
```

```
    %basicInlineTiming.attrib;
    %minMaxTiming.attrib;
    %Fill.attrib;
  ">

  <!ENTITY % timeContainer.attrib "
    %basicInlineTiming.attrib;
    %basicTimeContainers.attrib;
    %minMaxTiming.attrib;
  ">

  <!ENTITY %  par.content "%timeContainer.content;">
  <!ENTITY %  seq.content "%timeContainer.content;">

  <!ENTITY %  par.attrib "
    %timeContainer.attrib;
    %System.attrib;
  ">
  <!ENTITY %  seq.attrib "
    %timeContainer.attrib;
    %System.attrib;
  ">

  <!-- ================== Content Control ================================= -->
  <!ENTITY % BasicContentControl.module "INCLUDE">
  <!ENTITY % SkipContentControl.module  "INCLUDE">

  <!ENTITY % switch.content
   "((%media-object;|%link;)*|(%timeContainer;)*|layout)">

  <!-- ================== Layout ====================================== -->
  <!ENTITY % BasicLayout.module "INCLUDE">

  <!ENTITY % layout.content "(root-layout?,(region)*)">
  <!ENTITY % region.content "EMPTY">
  <!ENTITY % rootlayout.content "EMPTY">
  <!ENTITY % region.attrib "%skipContent.attrib;">
  <!ENTITY % rootlayout.attrib "%skipContent.attrib;">

  <!-- ================== Linking ===================================== -->
  <!ENTITY % LinkingAttributes.module "INCLUDE">
  <!ENTITY % BasicLinking.module      "INCLUDE">

  <!ENTITY % a.content      "(%media-object;)*">
  <!ENTITY % area.content   "EMPTY">
  <!ENTITY % anchor.content "EMPTY">

  <!ENTITY % a.attrib      "%smil-time.attrib;">
  <!ENTITY % area.attrib   "%smil-time.attrib; %skipContent.attrib;">
  <!ENTITY % anchor.attrib "%smil-time.attrib; %skipContent.attrib;">

  <!-- ================== Media  ====================================== -->
  <!ENTITY % BasicMedia.module    "INCLUDE">
  <!ENTITY % MediaClipping.module "INCLUDE">

  <!ENTITY % media-object.content "(area|anchor)*">
  <!ENTITY % media-object.attrib "
    %smil-time.attrib;
    %System.attrib;
    %Region.attrib;
  ">

  <!-- ================== Structure ================================== -->
  <!ENTITY % smil.content "(head?,body?)">
  <!ENTITY % head-layout.content     "layout|switch">
  <!ENTITY % head.content "(%head-layout.content;)?">
  <!ENTITY % body.content "(%timeContainer;|%media-object;|
                            %content-control;|%link;)*">

  <!-- ================== End of smilbasic-model-1.mod ================== -->



  <!-- ================================================================== -->
  <!-- SMIL 2.0 Basic DTD ================================================ -->
  <!-- file: SMIL20Basic.dtd

  This is SMIL 2.0 Basic, a proper subset of SMIL 2.0.

  Copyright 2000 World Wide Web Consortium (Massachusetts Institute of Technology, Institut National de Recherche en Informatique
  et en Automatique, Keio University. All Rights Reserved.

  Permission to use, copy, modify and distribute the SMIL Basic DTD and its accompanying documentation for any purpose and without
  fee is hereby granted in perpetuity, provided that the above copyright notice and this paragraph appear in all copies.  The
  copyright holders make
  no representation about the suitability of the DTD for any purpose.

  It is provided "as is" without expressed or implied warranty.
  Author:     Jacco van Ossenbruggen, Kenichi Kubota
  Revision:   $Id: SMIL20Basic.dtd,v 1.3 2000/09/21 11:16:46 jvanoss Exp $

  -->
  <!-- This is the driver file for the SMIL Basic DTD.

       Please use this formal public identifier to identify it:

           "-//W3C//DTD SMIL 2.0 Basic//EN"
  -->

  <!ENTITY % NS.prefixed "IGNORE" >
```

```
<!ENTITY % SMIL.prefix "" >

<!-- Define the Content Model -->
<!ENTITY % smil-model.mod
    PUBLIC "-//W3C//ENTITIES SMIL 2.0 Basic Document Model 1.0//EN"
           "smilbasic-model-1.mod" >

<!-- Modular Framework Module ..................................... -->
<!ENTITY % smil-framework.module "INCLUDE" >
<![%smil-framework.module;[
<!ENTITY % smil-framework.mod
     PUBLIC "-//W3C//ENTITIES SMIL 2.0 Modular Framework 1.0//EN"
            "smil-framework-1.mod" >
%smil-framework.mod;]]>

<!--    The SMIL 2.0 Basic Profile supports the lightweight multimedia
        features defined in SMIL language. This profile includes
        the following SMIL modules:

             SMIL 2.0 BasicLayout Module
             SMIL 2.0 BasicLinking Module
             SMIL 2.0 BasicMedia and MediaClipping Modules
             SMIL 2.0 Structure Module
             SMIL 2.0 BasicInlinTiming, SyncbaseTiming, EventTiming,
                      MinMaxTiming and BasicTimeContainers Modules
             SMIL 2.0 BasicContentControl and SkipContentControl Modules
-->

<!ENTITY % layout-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Layout//EN"
  "SMIL-layout.mod">
%layout-mod;

<!ENTITY % link-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Linking//EN"
  "SMIL-link.mod">
%link-mod;
<!ENTITY % BasicLinkingModule "INCLUDE">

<!ENTITY % media-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Media Objects//EN"
  "SMIL-media.mod">
%media-mod;

<!ENTITY % struct-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Document Structure//EN"
  "SMIL-struct.mod">
%struct-mod;

<!ENTITY % timing-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Timing//EN"
  "SMIL-timing.mod">
%timing-mod;

<!ENTITY % control-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Content//EN"
  "SMIL-control.mod">
%control-mod;
```

References

[SMIL1] "Synchronized Multimedia Integration Language (SMIL) 1.0 Specification", Philipp Hoschka, editor, 15 June 1998, available at http://www.w3.org/TR/REC-smil/

[SMIL2] "Synchronized Multimedia Integration Language (SMIL) 2.0 Specification", Jeff Ayars et al, editors, 05 June 2001, available at http://www.w3.org/TR/smil20/

[MOBILE-GUIDE] "HTML4.0 Guidelines for Mobile Access", T. Kamada, Takuya Asada, Masayasu Ishikawa, Shin'ichi Matsui. W3C Note 15 March 1999, available at http://www.w3.org/TR/NOTE-html40-mobile

[XHTML-BASIC] "XHTML Basic", Masayasu Ishikawa, Shinichi Matsui, Peter Stark, Toshihiko Yamakami. W3C Recommendation 19 December 2000, available at http://www.w3.org/TR/xhtml-basic/

[WAI-SMIL-ACCESS] "Accessibility Features of SMIL " Marja-RiittaKoivunen, Ian Jacobs. W3C NOTE 21 September 1999, available at http://www.w3.org/TR/SMIL-access

[CC/PP] "CC/PP", technology of the W3C Mobile Interest Group. W3C Note 27 July 1999, available at http://www.w3.org/TR/NOTE-CCPP/

[XML10] "Extensible Markup Language (XML) 1.0" T. Bray, J. Paoli and C.M. Sperberg-McQueen. W3C Recommendation, 10 February 1998, available at http://www.w3.org/TR/REC-xml

On-Demand Authoring (ODA) Studio Design Specification

Document Number: xxx-xxxx-xx
Version Number: x.xx
Document File Name: ODA Design Specification
Date: March 7, 2001
Authors: Xxxxxx, xxxxxx,
Formats: PDF, HTML, DOC, FMK?
Location: \\xxxxxxxx\c:\xxxxxxxxx.doc
Online Location: http://xxxxx/xxxxx/xxx.pdf
Review Process

Changes:	Date (Submitted)	Date (returned)	Approved (Initials)
Updated to reflect newest changes		March 9, 2001	

Publication History

Version #	Date	Main Changes
1.00	February, 2001	Created Document

Related Documents

Document #	Document Name
xxx-xxxx-xx	Xxxxxxxxx

Executive Summary

Table of Contents

On-Demand Authoring (ODA) Studio 1
Design Specification 1
Review Process 2
Publication History 2
Related Documents 2
Executive Summary 3
Table of Contents 4
1. Introduction 6
1.1. Purpose 6
1.2. Audience 6
1.3. Scope 6
1.4. Definitions, Acronyms, and Abbreviations 6
2. System Overview 7
2.1. Explanation 7
3. Design Considerations 7
3.1. Assumptions and Dependencies 7
3.2. Related Software or Hardware 7
3.3. General Constraints 7
3.4. Goals and Guidelines 7
3.5. Development Methods 7
4. Architectural Strategies 8
4.1. System Architecture 8
4.2. Programs which can be created in the ODA Studio 9
4.2.1. Type one: 9
4.2.2. Type two: 9
4.2.3. Type three: 9
4.2.4. Type four: 9
4.3. ODA Studio Container and ActiveX Controls 9
4.3.1. Waveform Editor 10
4.3.1.1. Recording 10
4.3.1.2. Basic Editing 10
4.3.1.3. Scrolling and Selecting 10
4.3.1.4. Segmentation 10
4.3.1.5. Copying and Pasting 10
4.3.1.6. Playback 10
4.3.2. Content Explorer 11
4.3.3. Prompter 11
4.3.3.1. Basic Features 11
4.3.4. Composer 11
4.3.4.1. Basic Features 11
5. Subsystem Architecture 12
5.1. Composer Component 13
5.1.1. Single Level program presentation 12
5.1.2. Navigation View 13
5.1.3. Multimedia layer view 13
5.1.4. Compression View 13
6. Skin capabilities 13
6.1. User Interface images (bitmaps) requirements for ODA Studio 13
6.1.1. Requirements for the Composer: 13
6.1.1.1. For the Prime segment: 13
6.1.1.2. For the Compound Segment: 13
6.1.1.3. ToolBar bitmap with an image for the following functions: (ComToolBar.bmp) 14
6.1.1.4. Icons for Multimedia layer view: 14
6.2. Requirements for the Content Explorer: 15
Background image - ConBk.bmp 15
6.2.1. Requirements for Wave Editor: 15
7. Detailed System Design 16
7.1. Basic design properties 16
7.2. Waveform Editor 16
7.2.1. Interfaces 16
7.2.2. Ambient Property Handling 17

7.2.3. Custom Properties 17
7.2.4. Drawing 17
7.2.5. User Interaction 17
7.2.5.1. Target/Source Panes 17
7.2.5.2. Editing Tools 17
7.2.5.3. Setting Markers 18
7.3. Content Explorer Control 18
7.3.1. Drag and Drop Mechanism 18
7.3.2. User Interaction 18
7.4. Prompter Control 19
7.4.1. Interfaces 19
7.4.2. User Interaction 19
7.5. Composer 19
7.5.1. Interfaces 19
7.5.2. User Interaction 20
8. Detailed Subsystem Design—D.A.V.I.D Systems 20
8.1. Overview 20
8.2. Composer ActiveX 20
8.3. Incoming Interface 21
8.3.1. Description 21
8.4. Outgoing Interface 22
8.4.1. Description 22
8.5. Client 23
8.5.1. Description 23
9. Glossary 23
10. Bibliography 23

1. Introduction
1.1. Purpose
The purpose of this document is to establish functional requirements and performance specifications for the ODA Studio
1.2. Audience
The intended audience for this document is Command Audio's management, operations, engineering, and software programming staff. This document will also serve as a basis of agreement between these parties as to the system's functionality and performance levels
1.3. Scope
The scope of this document will include constraints, functional requirements, and performance specifications.
1.4. Definitions, Acronyms, and Abbreviations
Audio-On-Demand Service A service that enables listeners to choose the programming they want to hear, whenever they want and wherever they are.
Block Indicates the type of compression in an audio layer

CAML Command Audio Markup Language (a derivative of XML)
CAML object
CAML element: The basic program building block in the ODA Studio Composer. An element can be a segment, layer, or block.
CAML header element CAML header elements allow introductory details of a program to be placed separately from the main story
CAML detail element: CAML detail elements allow additional content relating to header elements to be placed in this area.
Compound Segment A segment which contains more than one segment
Content Material fortransmission to Command Audio-enabled devices.
Layer Represents multiple types of media, such as audio, text, image or video
Program Playable content made up of elements
Prime Segment Smallest navigational unit, made up of one or more layers

CA Command Audio Corporation
ODA Studio On-demand Authoring Studio
2. System Overview
2.1. Explanation
The On-Demand Authoring (ODA) Studio is an editing software responsible for creating On-Demand Interactive Audio. Individuals use ODA to produce programs in a way that enables listeners to choose their programming and the specific stories within those programs which interest them.

The ODA Studio is created with the express intention of creating these navigable programs.

Single level and two-level programs can be created in the ODA studio. Single level programs provide fewer navigation opportunities than two-level programs. Two-level programs may include headers: the introductions to various stories within the program, and details: the specific stories which relate to the headers.

3. Design Considerations
3.1. Assumptions and Dependencies
? The ODA Studio runs on the Windows 2000 Operating System
? The ODA Studio should require minimal modifications for each system in which it is installed
? End users will include Command Audio Production Staff, as well as Production Staff or Broadcasters at Command Audio-partnered companies. End users will have the capability to put together programs from multimedia sources for transmission to Command Audio-enabled devices.
? The ODA Studio is not designed as a comprehensive replacement for other editing programs on the market, but instead is developed solely as a vehicle for creating Command Audio Content.
3.2. Related Software or Hardware
3.3. General Constraints
? For optimum editing, wave file size should not exceed XXMb
? Files in Content Explorer should be protected so that only one user may change contents at any one time.
3.4. Goals and Guidelines
? Should easily adapt to other partners' software
3.5. Development Methods
? Written in C++
? Once it was decided that the ODA Studio would operate on a Windows platform, Active X controls were chosen to distinguish the functional capabilities of each part of the studio.

4. Architectural Strategies
4.1. System Architecture

The ODA Studio was conceived as a replacement for the current Command Audio editing system, ENCO. One of the main benefits of the ODA Studio is the flexibility it offers. The ODA studio includes 4 components:

? Content Explorer displays multimedia files which are used to create a program

? Composer provides a visual stage to compose content. The Composer is made up of a Navigation View, Compression View, and Multimedia View.

? Prompter text content can be created and existing text files can be opened.

? Waveform Editor is the place where audio files are edited and segmented into On Demand Interactive Audio.

? segment audio programs specifically for Command Audio-enabled receivers.

4.2. Programs which can be created in the ODA Studio
4.2.1. Type one:
A program which requires minimal processing, may only need to be marked for compression. It is comprised of one prime segment only but contains at least one media layer. For example, the prime segment could contain a music track, associated textual information about the singer and writer, and an image of the singer. There are no navigational elements in a prime segment.
4.2.2. Type two:
A single level program which can be linked in such a way that a listener can navigate between different segments. Each of the segments is a prime segment capable of having several media layers. The navigation is sequential. For example, a listener can listen to story 1, skip stories 2 and 3, listen to story 4, and go back to hear story 2 by scanning back through 3.
4.2.3. Type three:
A two level program which allows additional navigation options. The program consists of a sequence of headers with associated detail segments. The user can now navigate from header to header or can navigate from a header to the associated detail. Instead of only being able to navigate between segments, a listener can navigate story headlines (headers) before deciding if they are interested in hearing the details behind the headlines. These programs allow a listener to navigate back and forth through headers and details in the program segments
4.2.4. Type four:
The fourth type of program is a type three program offering the user even more navigational options. The user can essentially navigate even within a header or a detail segment. The segments are marked in such a way that the listener can navigate within the headers and details. If there are two separate thoughts in the headline, these might be marked in such a way that the listener could skip, forward, rewind within headers. The segments are marked in such a way that the listener can hear/choose specific portions of the details as well.
4.3. ODA Studio Container and ActiveX Controls
The ODA Studio Container hosts 4 ActiveX controls:

? Content Explorer Control displays the content required to build the output multimedia CAML objects, and provides the ability to drag and drop these objects into other parts of the ODA Studio. Examples of this content include: audio programs, earcons, texts, images etc.
? Prompter Control is the ActiveX component that allows text related to a specific audio recording to be created or edited
? Waveform Editor Control is responsible for multiple functions of editing audio files (wave files)
? Composer Control provides functionality for composing a CAML multimedia object from multiple source components.
4.3.1. Waveform Editor
Presently, Waveform Editor Control is implemented within ATL framework. It utilizes OLE Drag And Drop for exchanging wave files with other components. It uses DirectSound API for accessing the system's audio resources. MFC is used for rendering waveforms and GUI.

? Ability to have two playback modes. In Mode 1, cursor moves across the waveform in sync as the waveform scrolls forward. In Mode 2, waveform view does not scroll forward when cursor reaches right edge, but playback continues.
4.3.1.1. Recording
? Ability to indicate recording is in process by illuminating the record button and peek meter
? Ability to pause and resume recording by selecting pause or play.
? Ability to save wave file by using windows keyboard shortcuts or file menu
4.3.1.2. Basic Editing
? Ability to rewind, play, stop, pause, forward, zoom in and zoom out using buttons on the toolbar
? Ability to undo last action (actions to be defined later) using a button on the toolbar or a keyboard shortcut like Ctrl Z
4.3.1.3. Scrolling and Selecting
? Ability to scroll left and scroll right using arrows on the keyboard or the mouse and scrollbar
? Ability to zoom the wave into at least 100th of a second, and zoom out to view the entire waveform using the zoom in/out buttons on toolbar.
? Ability to select sections of the wave using the mouse and ctrl key.
? Ability to adjust the selection by single pixels from the left or right border using keyboard arrows and the shift key
4.3.1.4. Segmentation
? Ability to insert/delete markers and compression blocks into imported waves by right clicking with the mouse
? Ability to keep markers consistent even if audio between them has been removed or changed.
4.3.1.5. Copying and Pasting
? Ability to copy and paste wave sections from the Source (recording) pane to the Target pane and within the Target Pane.
? Ability to replace selected portion of waveform with data from the internal clipboard at the insertion point by using Windows shortcuts.
4.3.1.6. Playback
? Ability to begin playback of a current selection or of the entire waveform from either the left edge of the waveform display or from the cursor position.

4.3.2. Content Explorer
Content Explorer is based on a MFC CTreeView class that simplifies the usage of the tree control CTreeCtrl, the class that encapsulates tree-control functionality.

Content Explorer is a MFC ActiveX control. Its two objectives:
? To display (in tree/branches format) media sources available for building the output multimedia CAML objects. These sources can be audio programs, texts, images, commercials, etc. These building blocks can be files or records in databases.
? To initiate a dragging mechanism when moving or copying tree items to other parts of application.
4.3.3. Prompter
The Prompter ActiveX control provides text content for CAML objects. Prompter is a MFC ActiveX control based on a CRichEditView class. CRichEditView provides the functionality of the rich edit control within the context of MFC's document view architecture. CRichEditView maintains the text and formatting characteristic of text.
4.3.3.1. Basic Features
? Ability to flag parts of text for re-evaluation later in editing process. A keyboard symbol and shortcut needs to be assigned to this feature so that PS can easily doublecheck and review work. Flag will remain in prompter as long as any of these remain in txt. PS can place notes specific to file as well.
? Ability to differentiate comment text from script text using separate color and font and brackets. Announcer needs comment text to indicate pronunciations and emphasized words in script.
? Ability to create headers and details which pertain to waveform recording by pressing Control H and Control D.
? Ability to adjust font size and type using Windows keyboard shortcuts
? Ability to open content from content explorer and outside applications.
? Ability to save content which has been created in prompter, as well as content which has been imported and changed in prompter using Windows keyboard shortcuts.
? Ability to scroll down through text using up and down arrows on the keyboard or a scrollbar

4.3.4. Composer
Composer control provides functionality for assembling a CAML multimedia object from multiple source components. Composer control interacts with all other ActiveX controls. Its GUI depicts a CAML multimedia object composed of CAML elements. Currently a CAML element from GUI point of view is shown as a rectangle, and can be a prime segment or a compound segment.
4.3.4.1. Basic Features
? Segments can be inserted, deleted and or appended using the buttons on the toolbar.

? When the number of segments reaches the edge of the Composer View, you should be able to scroll forward through the view.

Segments can be viewed in any order, moving back and forth or up and down between headers and details by using the arrow keys on the keyboard

5. Subsystem Architecture
5.1. Composer Component

A new GUI design is implemented for 1.2 release. New features include:
5.1.1. Single Level program presentation
Before creating a program, choose the type of program you wish to create. They can select a single level or a two-level program from an options menu, and the appropriate screen will appear. To create programs which do not require complex navigation capabilities, choose a single-layer program. For more complex programs, two-level programs should be created. These programs will be segmented so that listeners will still be able to navigate between stories.
5.1.2. Navigation View
Files can be dragged and dropped, inserted, deleted, attached and appended assembled in the Navigation view. Because they are parts of the overall program composition, segments, layers and blocks will all be managed within the navigation view.
5.1.3. Multimedia layer view
You can select individual multimedia layers in the multimedia viewer. A preview pane is available to examine text, image, or video layers. A wave layer can still be viewed in the Waveform Editor. You can right click on a segment to view the properties of that segment. Properties can include:
? File name
? File size
? File length
? Date file was originally recorded
? Name of person who last worked on file and when
? Source of the original recording (satellite from_____, ISDN from _____, etc.)
? Sample rate at which it was recorded.
5.1.4. Compression View
The compression view displays the blocks which represent compression in a wave file. The compression view works with all parts of the multimedia viewer to illustrate the way the different layers fit together in a program.

6. Skin capabilities
6.1. User Interface images (bitmaps) requirements for ODA Studio
The targeted resolution for the ODA Studio is 1024 x 768 pixels. The composer on the upper right hand side of the studio should measure 600 x 330 pixels. The user will be able to choose the type of skin from a list of available skins. The properties of the skin can be applicable to one component or to all of them. Skin elements must be able to be stretched without distorting original image.
6.1.1. Requirements for the Composer:
1. A background image - ComBk.bmp
6.1.1.1. For the Prime segment:
? Prime segment empty selected - PriSegES.bmp
? Prime segment empty unselected- PriSegEUs.bmp
? Prime segment full selected - PriSegFUs .bmp
? Prime segment full unselected - PriSegFUs.bmp
? Detail: Prime segment empty selected - PriSegES(det).bmp
? Detail: Prime segment empty unselected- PriSegEUs(det).bmp
? Detail: Prime segment full selected - PriSegFUs(det) .bmp
? Detail: Prime segment full unselected - PriSegFUs(det).bmp
6.1.1.2. For the Compound Segment:
? A bitmap image that looks good when stretched.
? Compound segment selected - ComSegS .bmp (should this be ComSegES.bmp)
? Compound segment unselected- ComSegUs.bmp
? Compound segment full selected - ComSegFS.bmp
? Compound segment full unselected - ComSegFU.bmp
? Detail: Compound segment selected - ComSegS(det).bmp (ComSegES.bmp)
? Detail: Compound segment unselected- ComSegUs(det).bmp
? Detail: Compound segment full selected - ComSegFS(det).bmp
? Detail: Compound segment full unselected - ComSegFU(det).bmp
? Prime segment empty selected - PriSegES.bmp
? Prime segment empty unselected- PriSegEUs.bmp
? Prime segment full selected - PriSegEUs.bmp
? Prime segment full unselected - PriSegFUs.bmp
6.1.1.3. ToolBar bitmap with an image for the following functions: (ComToolBar.bmp)
? Append
? Attach File
? Delete
? Insert Before
? Insert After
? Assemble
? Player
6.1.1.4. Icons for Multimedia layer view:
? Icons need to be 16 X 16 pixels in size.
? One background image- ComLayBk
? For Segment (This is in place of a "+" sign in the tree view of the layer view. The tree view shows the layers in the segment)
? Opened segment - ComLaySegO.bmp
? Closed Segment- ComLaySegC.bmp

? For wave file- ComLayWav.bmp
? For Text File - ComLayTxt.bmp
? For Image file - ComLayImg.bmp
? For Video File - ComLayVid.bmp
? For Animation - ComLayAni.bmp
? For Compression view:
? For background - ComCompBk.bmp
? To represent different compressions like DVSI and DOLBY in the compression view we need two more bitmaps which will look good when resized:
? To represent DVSI - ComCompDVSI.bmp
? To represent DOLBY - ComCompDOL.bmp

6.2. Requirements for the Content Explorer:
Background image - ConBk.bmp
? Icon for the main repository
? Icons need to be 16 X 16 pixels in size.
? The following subfolder icons (We can either have a separate icon for each folder or a uniform icon for all folders).

? If we have only one icon for all folders:
? Con Fol C.bmp
? Con Fol O.bmp

? If we have separate icons for each type of folder:
? Audio Clips - ConFolAud.bmp

? Earcons - ConFolEar.bmp
? Music samples- ConFolMus.bmp
? Prompt files- ConFolPro.bmp
? Video files - ConFolVid.bmp
? Text Files- ConFolTxt.bmp
? Image Files- ConFolImg.bmp
? Wave Files—ConWav.bmp
? Text Files—Con Txt.bmp
? Music Files—Con Mus.bmp
? Animation Files—Con Avi.bmp
? Image Files—Con Img.bmp

The Icons created for Multimedia layer view can be used to denote each file in the Content Explorer also.
6.2.1. Requirements for Wave Editor:
Bitmap for toolbar of target pane - WavToolbarTrg.bmp, with the following buttons
? Rewind
? Play
? Stop
? Pause
? Fast forward
? Zoom in
? Zoom Out
? Scroll left
? Scroll right

Bitmap for toolbar of target pane - WavToolbarSrc.bmp, with the following buttons
? Rewind
? Play
? Stop
? Pause
? Record
? Fast forward
? Zoom in
? Zoom Out
? Scroll left
? Scroll right

Bitmap for navigating handle - WavNavHdl.bmp

7. Detailed System Design
7.1. Basic design properties
? The Container communicates with a control using a set of COM interfaces. Two-way communication is performed through connection points, advise sinks and other interfaces.
? The ODA Studio Container has ambient properties, which allow the container to provide information to its controls. This ensures a uniform look and feel for all the Controls visible inside the Container. The Controls are supposed to react to the change of the ambient properties and synchronize the appearance and behavior with the other controls accordingly.
? The ODA Studio Controls have properties which affect the behavior of a control. They can also be used to pass data to and from the control.
? The ODA Studio Controls also support stock properties, which are standard properties like Font and Color. Standard properties are also known to the Container and therefore can be bound. Specifically, Control can inform Container that a stock property is about to change and allows the Container to reject the change.
? Control's Properties are persistent so that information is retained when control is closed. When control is restarted, it will default to the settings used when last closed.
? The ODA Studio Controls are required to respond to ambient property changes
7.2. Waveform Editor
7.2.1. Interfaces
Currently Waveform Editor Control exposes one interface -
IWaveEditObj with five methods:

SetWavePath(BSTR bstrPath): When a wave file is selected, the variable bstrPath passes the name of the wave file to the Waveform Editor.
Play(): Commands Waveform Editor to play loaded wave file.
StopPlay(): Commands Waveform Editor to stop playing loaded wave file.
GetSegmentCount(long * pnCount): Returns the number of segments in loaded wave file in variable pnCount.
GetSegmentPathAt(long index, BSTR * bstrPath): Creates wave file for the segment indicated by index and returns its path in bstrPath.

7.2.2. Ambient Property Handling
On startup the control calls CComControl's methods:
GetAmbientBackColor, GetAmbientForeColor,GetAmbientFont to set up stock properties.
When an ambient property changes, the Container calls
IOLEControl::OnAmbientPropertyChange()
on Waveform Editor Control passing DISPID of the property that has changed. The control overrides this method, obtains property value and handles the change.
7.2.3. Custom Properties
For setting up custom properties a property page is required.
Initializing properties
Persisting properties
7.2.4. Drawing
Synchronous scrolling is implemented to display a playing waveform.
Current rendering rate is 10 panes per second.
7.2.5. User Interaction
7.2.5.1. Target/Source Panes
Waveform Editor consists of two editing panes. In the target pane, extra material from the recorded wave can be removed. It for can also be marked for scanning and compression purposes. In the lower (source) pane, sound can be recorded through a mixer for inclusion into the wave file in the target pane.

Either pane can be accessed by clicking on the left mouse button. Wave files from other applications may be dropped into the target pane. The target pane can also be a drag source. Drag and drop is implemented using standard OLE drag and drop mechanism.

The source pane has the same functionality as the target pane, and also allows for sound recording through a mixer (there is a "record" button in play control). Selections can be dragged outside of the control. Selections can also be dragged to the target pane for inclusion into an existing wave file loaded there (not implemented yet).

The target and the source panes are separated by a splitter, which allows the pane windows to be re-sized.
7.2.5.2. Editing Tools
Waveform Editor has five control buttons: "go to start", "play", "pause" (not implemented), "stop", "go to end". NB: "record" button is absent in the target pane.

The target pane also has four control buttons: "zoom in", "zoom out", "scroll back", and "scroll forward". Scrolling is also possible with a horizontal scroll bar at the top of the pane.

The pane has horizontal rulers to measure the playback time up to a 100th of a second. The rulers change scale with window resizing and scroll in sync with playback.
7.2.5.3. Setting Markers
A navigation handle indicates the current position in the wave file. Handle can be moved back or forward with the mouse. Sound will be played when handle is moved using mouse (scrubbing).

Markers can be set to define compression blocks. Markers have the following attributes: compression (Dolby, DVSI, Skip). The marker attributes are indicated in a dialog window. Segment markers have labels to indicate compression choice. Markers can subsequently be removed by double clicking on segment marker and pressing "Remove marker" button.

Portions of the wave file can be selected by holding the ctrl key down while dragging the left mouse button. Selections are draggable outside of the control.

The Context menu appears when right mouse button is clicked. Menu options include setting segment marker at current mouse position, selection cut, copy and paste (not implemented).
7.3. Content Explorer Control
7.3.1. Drag and Drop Mechanism
Content Explorer implements a MFC OLE drag-and-drop mechanism. When a user transfers data, using either the Clipboard or drag and drop, the data has a source and a destination. Content Explorer provides the data for copying and another part of application (it can be Composer or Waveform Editor) accepts it for pasting. The MFC provides two classes that represent each side of this transfer:

Data source (as implemented by COleDataSource object) represents the source side of the data transfer. It is created by the source Content Explorer when data is to be copied to the Clipboard, or when data is provided for a drag-and-drop operation.

Data object (as implemented by COleDataObject object) represents the destination side of the data transfer. It is created when the destination ActiveX has data dropped into it, or when it is asked to perform a paste operation from the Clipboard.

Currently a well-known CF_HDROP format is used for dragging and dropping files.
7.3.2. User Interaction
Content Explorer displays all folders and files that reside in the directory called Repository. A user can drag a tree item and drop it onto a layer, segment, tree view or waveform editor. Text files can be dragged file into Prompter.
7.4. Prompter Control
7.4.1. Interfaces
The Prompter ActiveX control has just one _DPrompter interface.

To get a pointer to this interface:
```
CComPtr<_DPrompter> pObj;
pPrompterCtrl->QueryInterface(__uuidof(_DPrompter),  (LPVOID *)&pObj);
pObj->FooMethod();
```

Here pPrompterCtrl of type LPOLEOBJECT is a pointer to the Prompter ActiveX control .

The _DPrompter interface has the following methods:
```
void OnFileOpenText();
void OnFileSaveText();
void OnFileSaveTextAs();
BSTR GetDocumentPath();
BSTR GetContent();
BOOL IsModified();
```
7.4.2. User Interaction
A User can do the following things:
? Open the text file from File menu and select the "Open Prompter Text" menu item.
? OR, place cursor in Prompter and begin composing text.
? Edit the text.
? Text can be divided into headers and details. To insert a header marker, a user must press Ctrl and H keys at the same time. To insert a detail marker, a user must press Ctrl and D keys at the same time. Prompter automatically numbers headers and details sequentially. For example, if the last header is <Header> 5, pressing Ctrl and H keys puts <Header> 6 into the cursor position.
? Save the text using one of the following: Save As, Save Prompter Text, or Save Prompter Text As, from the File menu.
7.5. Composer
7.5.1. Interfaces
Currently the Composer ActiveX control has just one _DComposer interface. The following code shows how to get this interface:
```
CComPtr<_DComposer> pObj;
pComposerCtrl->QueryInterface(__uuidof(_DComposer), (LPVOID *)&pObj);
pObj->FooMethod();
```
Here pComposerCtrl is a pointer of class LPOLEOBJECT to the Composer ActiveX control.
Currently this interface has just one method:
SetPrompter((LPUNKNOWN*)&(pPrompter)). The Composer ActiveX control has a member of class ColeClientItem* m_pPrompter. This method sets m_pPrompter to the Prompter ActiveX control pointer.
7.5.2. User Interaction
Composer has the following buttons:
Append. Appends new empty header and detail elements to the CAML object.
Attach. Opens a file dialog box. A user can choose a file and attach it to the selected element.
Delete. Deletes a selected element. In a two-level program, if a header is selected, its corresponding detail element will automatically be deleted as well.
Insert before. Inserts segment before the selected element. In a two-level program, an upper and lower element will be inserted.
Insert after. Inserts segment after the selected element. In a two-level program, an target and lower element will be inserted.
Play next. Plays next header element
Play previous. Plays previous header.
Play down. Plays the header's corresponding detail element when a header element is selected
Play up. Plays the detail's corresponding header when a detail element is selected.
Stop. Stops playing a selected element.
Assemble. Attaches wave segments from the source pane of the Waveform Editor control to the CAML elements according to the prompter text layout. The first segment is assigned to the first header/detail in the prompter text, and so on.

In addition, a user can do the following things:
? Drag a wave file from the Content Explorer control or Microsoft Windows Explorer and drop it on a CAML element or multmedia layer in the Composer control.
? Drag a selected wave segment from the first pane of the Waveform Edit control and drop it on a CAML element.
? Move a file attached to a CAML element to another element.
? Append a file attached to a CAML element to a file attached to another element. To do it a user has to press a Ctrl key and move a file at the same time.

8. Detailed Subsystem Design-D.A.V.I.D Systems
8.1. Overview
The On-Demand Authoring Center's Composer ActiveX (control) interacts with the Multitrack Editor (client). The Multitrack Editor is a client of the Composer. Communication between the two is bi-directional: the client controls the Composer and vice versa.
8.2. Composer ActiveX

The Composer implements two kinds of interfaces: incoming and outgoing. The incoming interface contains methods that are called by the client. The outgoing interface is used to notify the client of the control's events. It is called a sink interface, because it sinks event notifications.

8.3. Incoming Interface

8.3.1. Description

Control's incoming interface is called _DComposer and is created by the MFC ActiveX Control Wizard. The current _DComposer interface has the following methods:

```
- void    BSTR GetBlocks (BSTR bstrObjectID)
- void    long OnDelete (BSTR bstrObjectID);
- void    long OnDragDrop (BSTR bstrFormat, long X, long Y, short mode, BSTR bstrObject);
- void    OnFileNew ();
- void    BSTR OnGetMetadata();
- void    long OnSetMetadata (BSTR bstrComposerMetadata);
- void    long SetProperties (BSTR bstrSetProperties);
```

The GetBlocks (BSTR bstrObjectID)method returns an XML string containing information about blocks in the object with a given object ID.

The OnDelete (BSTR bstrObjectID) method is called by the client when an object is being deleted.

Clips from the client multi-format clipboard or any track can be moved to a composer element (header or detail prime segment). The OnDragDrop(…) method is repeatedly called by the client during drag-and-drop operations. The method has the following parameters:

- bstrFormat represents a drag format which is an XML string with the following structure:

```
- <objectinfo>
<objectid>objectid</objectid>
<dur>duration in US</dur>
- </objectinfo>
```

X and Y are the current mouse coordinates. Using the coordinates, the Composer can determine whether the mouse is inside the particular Composer element;

- mode indicates whether drag or drop operation occurs;
- bstrObject is a string representation of the objectid of the dragged object.

The OnFileNew( ) method is called to create a new project in the Composer.

The Composer Metadata presents the structure and content of the program loaded in the Composer. The client calls the OnGetMetadata() method to get the Composer Metadata. The return value is a string representation of the Composer Metadata. So, the serialized project includes this string. The Metadata string is an XML string.

The OnSetMetadata (BSTR bstrComposerMetadata) method is called by the client upon loading the project. The Composer, using bstrComposerMetadata, can restore the previously saved program. BstrComposer Metadata is an XML string.

SetProperties (BSTR bstrSetProperties) is used for previewing layers selected in Multimedia Layer View. The parameter bstrProperties is an XML string with the following structure:

```
<objectinfo>
          <objectid>  </objectid>
          <dur> </dur>
          <markerinfo>
                    <marker>
                              <pos> </pos>
                              <title> </title>
                              <type> </type>
                    </marker>
          </markerinfo>
 </objectinfo>
```

8.4. Outgoing Interface

8.4.1. Description

Control's outgoing interface is called _DcomposerEvents and is created by the MFC ActiveX Control Wizard. The current _DComposerEvents interface has the following events:

```
- void EditObject (long objectID, long*result)
- void GetProperties (BSTR bstr PropDescr)
- void PlayObject (long objectID, long*result)
- void StopPlayObject (long objectID, long*result)
- void SaveCurrentState(long*result)
- void SaveObject (long objectID, BSTR path, long*result)
- void SaveSource (BSTR SaveSourcePath, BSTR* bstrSourceName)
- void GetFile (BSTR xmlObjectInfo, long*result)
```

The client (not the control) implements all these events. The Composer can only fire the events. For example, to fire SaveObject event the Composer executes the following code:

FireEvent(dispidSaveObject,EVENT_PARAM(VTS_I4 VTS_WBSTR), objectID, path). FireEvent through the COM mechanism passes control to the client where the corresponding method is called.

The bstrPropDescr parameter of the GetProperties event is an XML string with the structure:

```
<objectinfo>
          <objectid> </objectid>
</objectinfo>
```

The GetFile parameter is an XML string with the structure:

```
<objectinfo>
          <objectid> </objectid>
          <path> </path>
          <format> </format>
</objectinfo>
```

8.5. Client

8.5.1. Description

The client does the following things:

? implements events that the Composer fires. To do this a class of type CAdviseSink that derives from _DcomposerEvents has been created

? launches the Composer ActiveX that allows the client to create the embedded OLE item of type CDavidClientCntrItem that derives from COleClientItem. To do this the client needs to know just the Composer ProgID "COMPOSER.ComposerCtrl.1"

? defines the Composer position in the client area

? gets a Composer pointer of type IConnectionPointContainer

? using the last interface finds the IConnectionPoint interface

? creates an object of type CAdviseSink

? through IConnectionPoint interface calls Advise method that passes the pointer to the CAdviseSink object
9. Glossary
10. Bibliography
1. Functional requirements

We Claim:

1. A method of editing audio or video data using a display, comprising the acts of:

providing a plurality of segments of the audio or video data;

displaying an icon representing each of the segments;

associating a first one of the segments with at least one other of the segments; and

displaying an indication of the association of the first to the other of the segments thereby composing a program of the segments;

wherein the program may be played in a sequence determined by a user in accordance with the associations.

2. The method of Claim 1, wherein the associations are each a hyperlink.

3. The method of Claim 1, wherein the associations are each displayed as a spatial arrangement of the icons.

4. The method of Claim 1, wherein each icon when activated results in playing at least a portion of the segment represented by the icon.

5. The method of Claim 1, wherein each icon carries an identifier for the segment represented by the icon.

6. The method of Claim 1, wherein each displayed icon is designated as a first type or a second type, a plurality of the second type being associated with each of the first type.

7. The method of Claim 1, wherein each displayed icon is designated as a first type or a second type, a plurality of the second type associated with each of the first type or the second type.

8. The method of Claim 1, each icon representing content selected from the group consisting of audio, video, image, or text.

9. The method of Claim 6, wherein the program when played by the user plays each of the segments represented by an icon of the first type in a predetermined order, and allows user selection of playing each of the segments represented by an icon of the second type.

10. The method of Claim 7, wherein the program when played by the user plays each of the segments represented by an icon of the first type in a predetermined order, and allows user selection of playing each of the segments represented by an icon of the second type.

11. A computer system having a graphical user interface for editing audio or video data on a display, comprising:

a memory storing a plurality of segments of audio or video data,

a portion assigning an icon to be displayed representing each of the stored segments;

a portion which associates a first one of the segments with at least one other of the segments, and displays an indication of the association;

a composer which composes a program of a set of the associated segments; and

wherein the composed programs may be played in a sequence determined by a user in accordance with the associations.

12. The system of Claim 11, wherein the associations are each a hyperlink.

13. The system of Claim 11, wherein the associations are each displayed as a spatial arrangement of the icons.

14. The system of Claim 11, wherein each icon when activated plays at least a portion of the segment represented by the icon.

15. The system of Claim 11, wherein each icon carries an identifier for the segment represented by the icon.

16. The system of Claim 11, wherein each displayed icon is designated as a first type or a second type, a plurality of the second type being associated with each of the first type.

17. The system of Claim 11, wherein each displayed icon is designated as a first type or a second type, a plurality of the second type associated with each of the first type or the second type.

18. The system of Claim 11, each icon representing content selected from the group consisting of audio, video, image, or text.

19. The system of Claim 16, wherein the program when played by the user plays each of the segments represented by an icon of the first type in a predetermined order, and allows user selection of playing each of the segments represented by an icon of the second type.

20. The system of Claim 17, wherein the program when played by the user plays each of the segments represented by an icon of the first type in a predetermined order, and allows user selection of playing each of the segments represented by an icon of the second type.

21. An improvement to a computer editing system, the system allowing composition of a program from a plurality of audio or video segments using a graphical user interface displaying an icon identifying each of the segments, comprising:

assigning links between the segments by displaying a spatial arrangement of the icons;

the spatial arrangement representing a link between a header segment and a detail segment wherein the program includes a plurality of header segments played in a predetermined order and a plurality of detail segments played in an order of election of a user in response to an indication of each link.

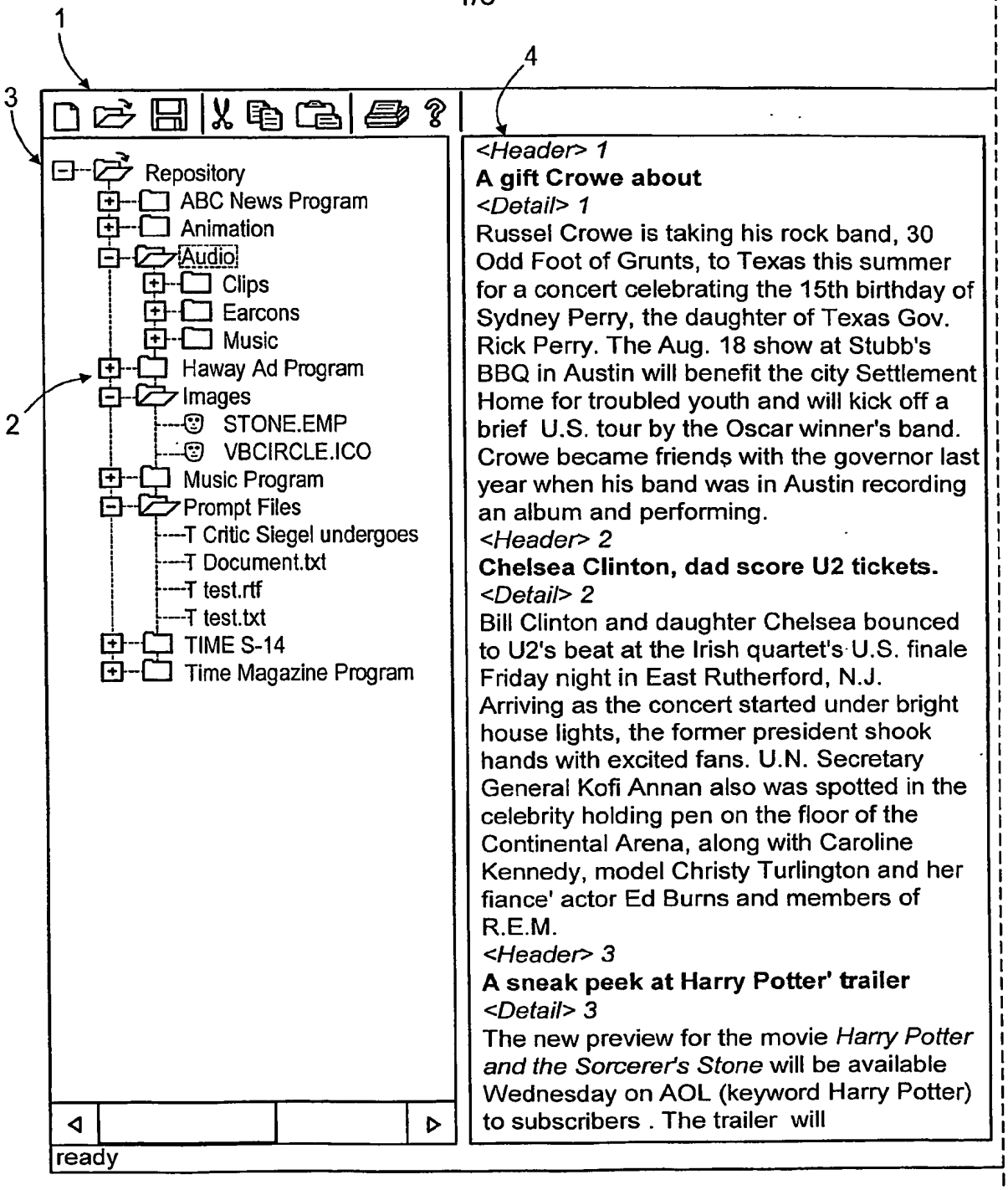


Fig. 1A

Fig. 1

Fig. 1A	Fig. 1B

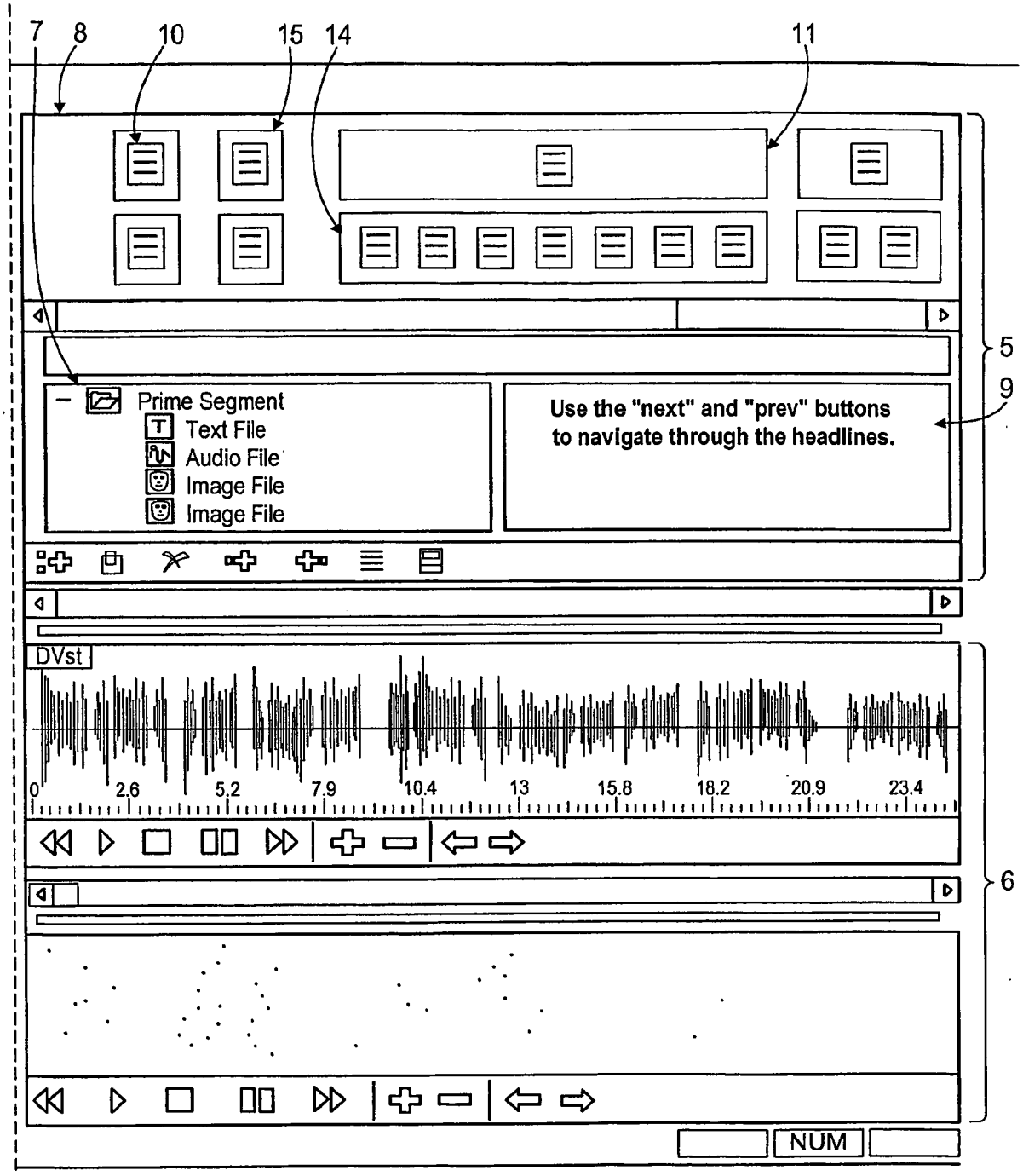

7
8
10
15
14
11
5
9
Prime Segment
Text File
Audio File
Image File
Image File
Use the "next" and "prev" buttons to navigate through the headlines.
DVst
0
2.6
5.2
7.9
10.4
13
15.8
18.2
20.9
23.4
6
NUM

Fig. 1B

Fig. 2A

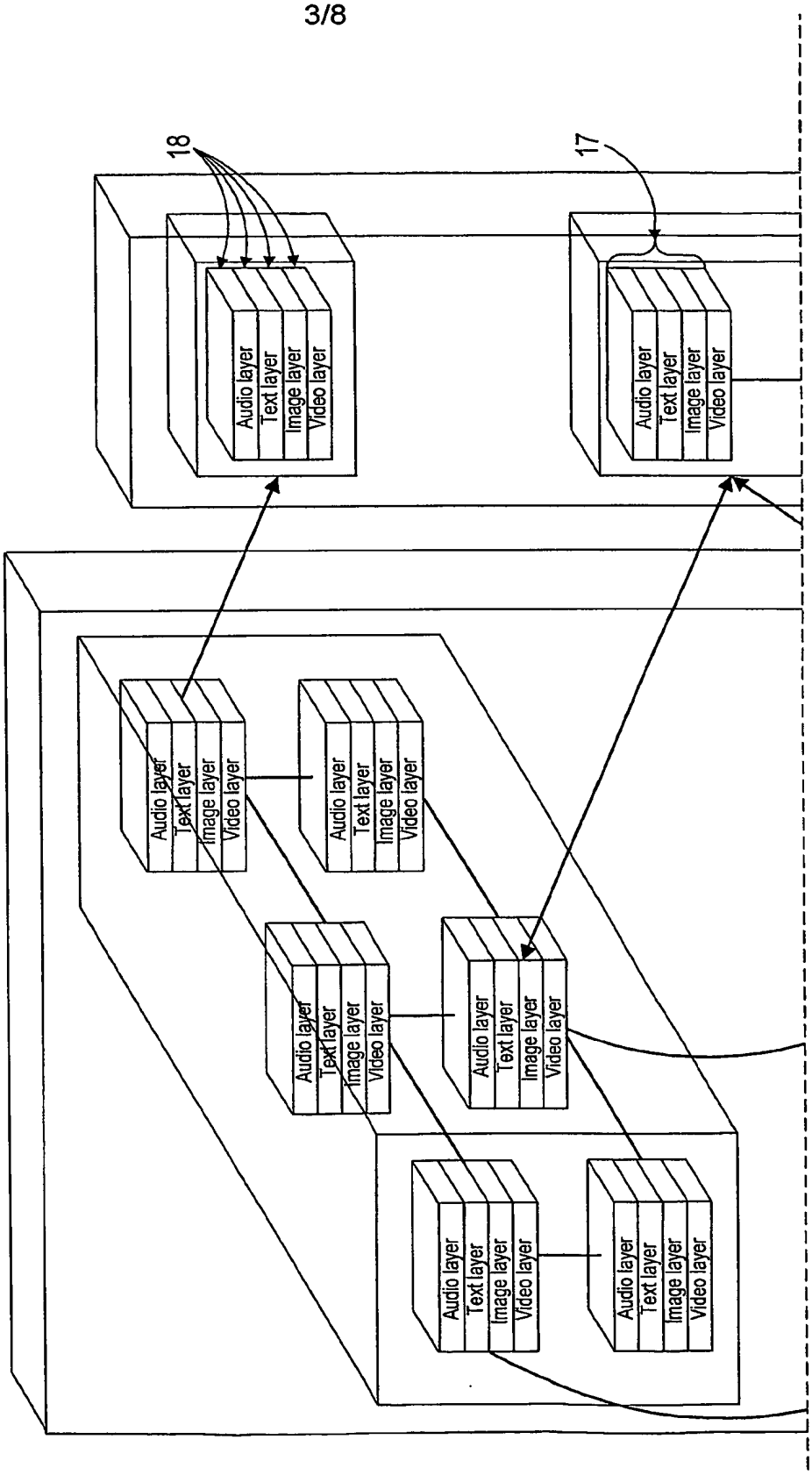


Fig. 2

Fig. 2A
Fig. 2B

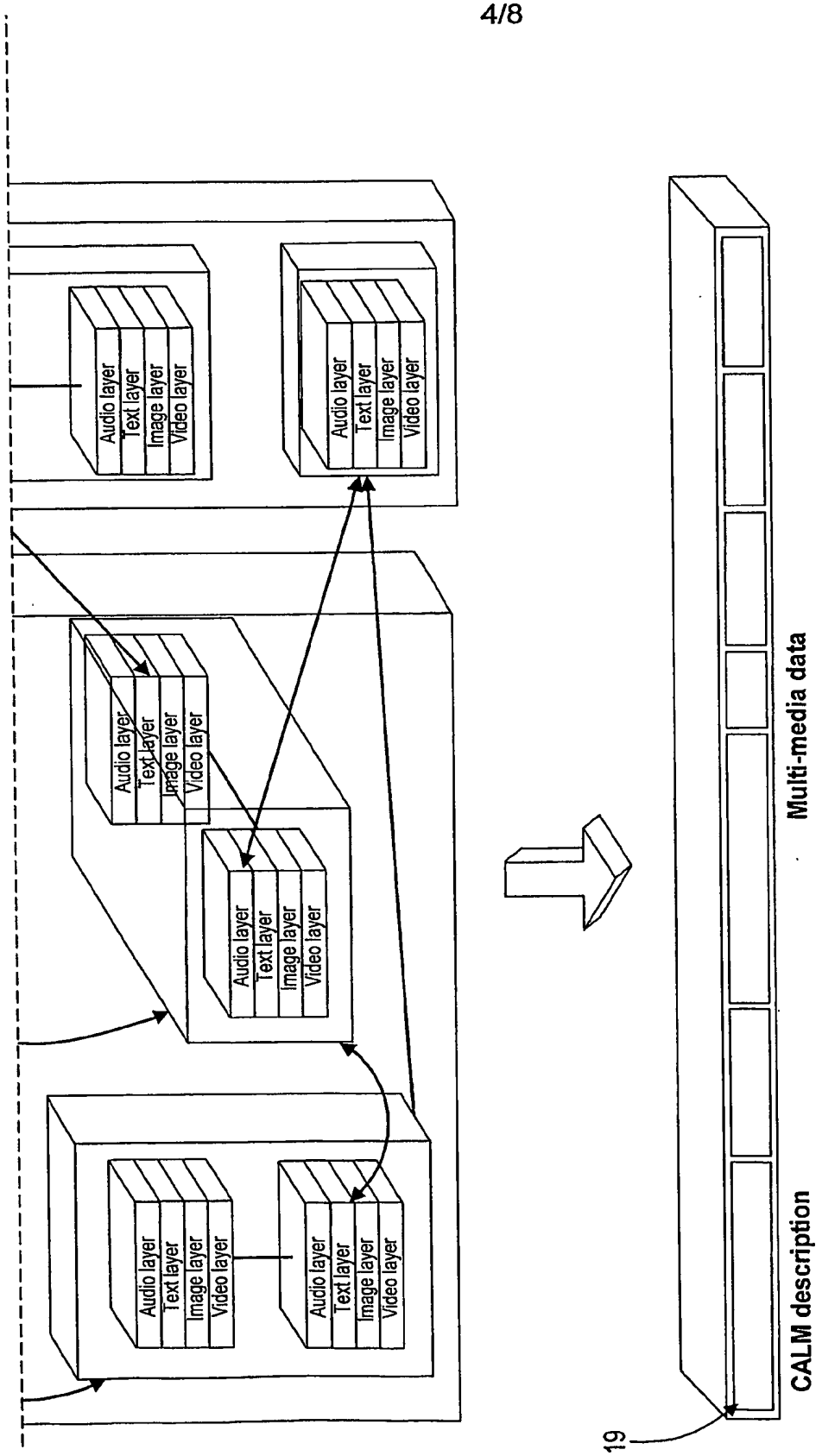

Audio layer
Text layer
Image layer
Video layer
Multi-media data
CALM description
19

Fig. 2B

Fig. 3

Fig. 3A | Fig. 3B

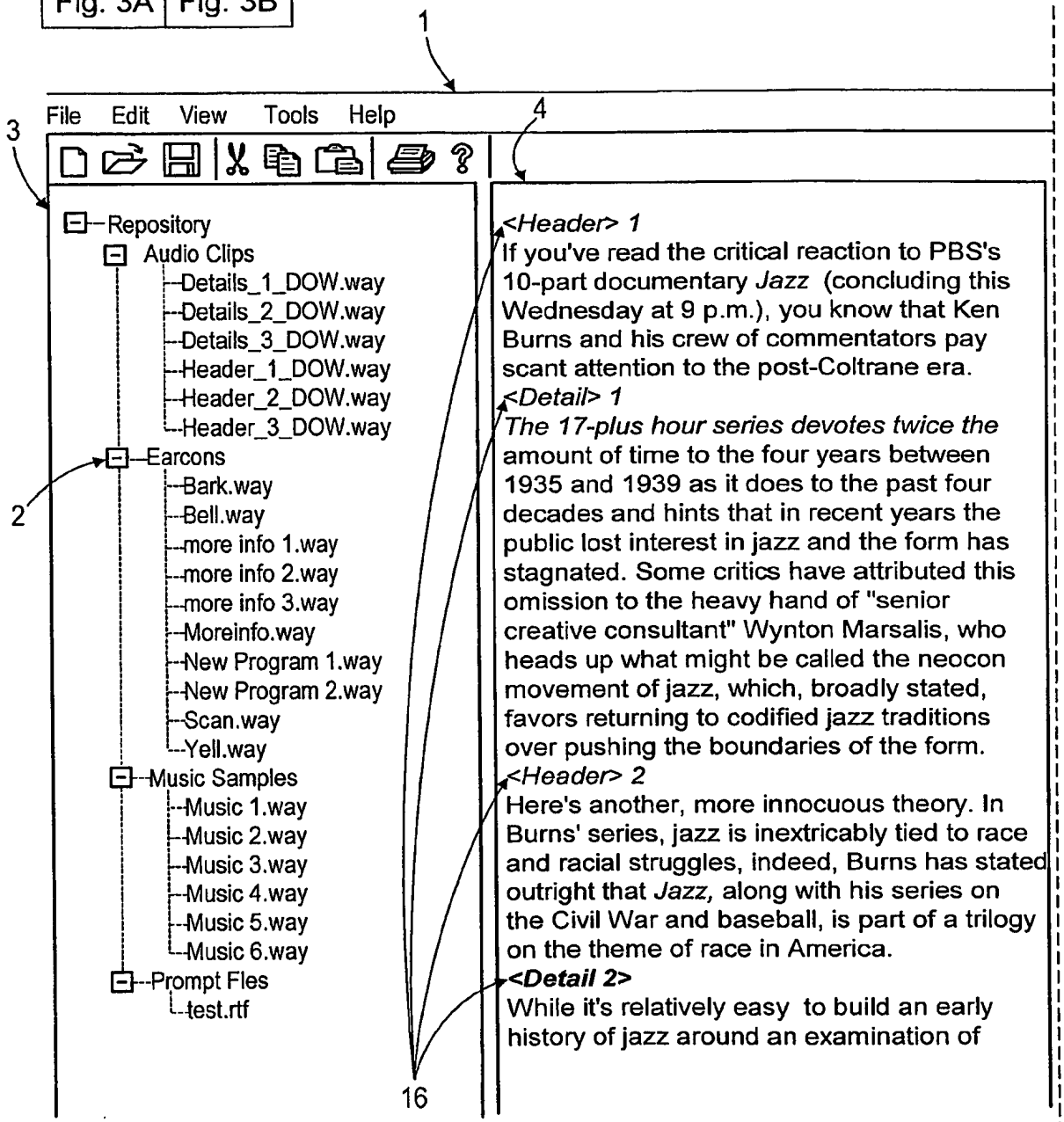


Fig. 3A

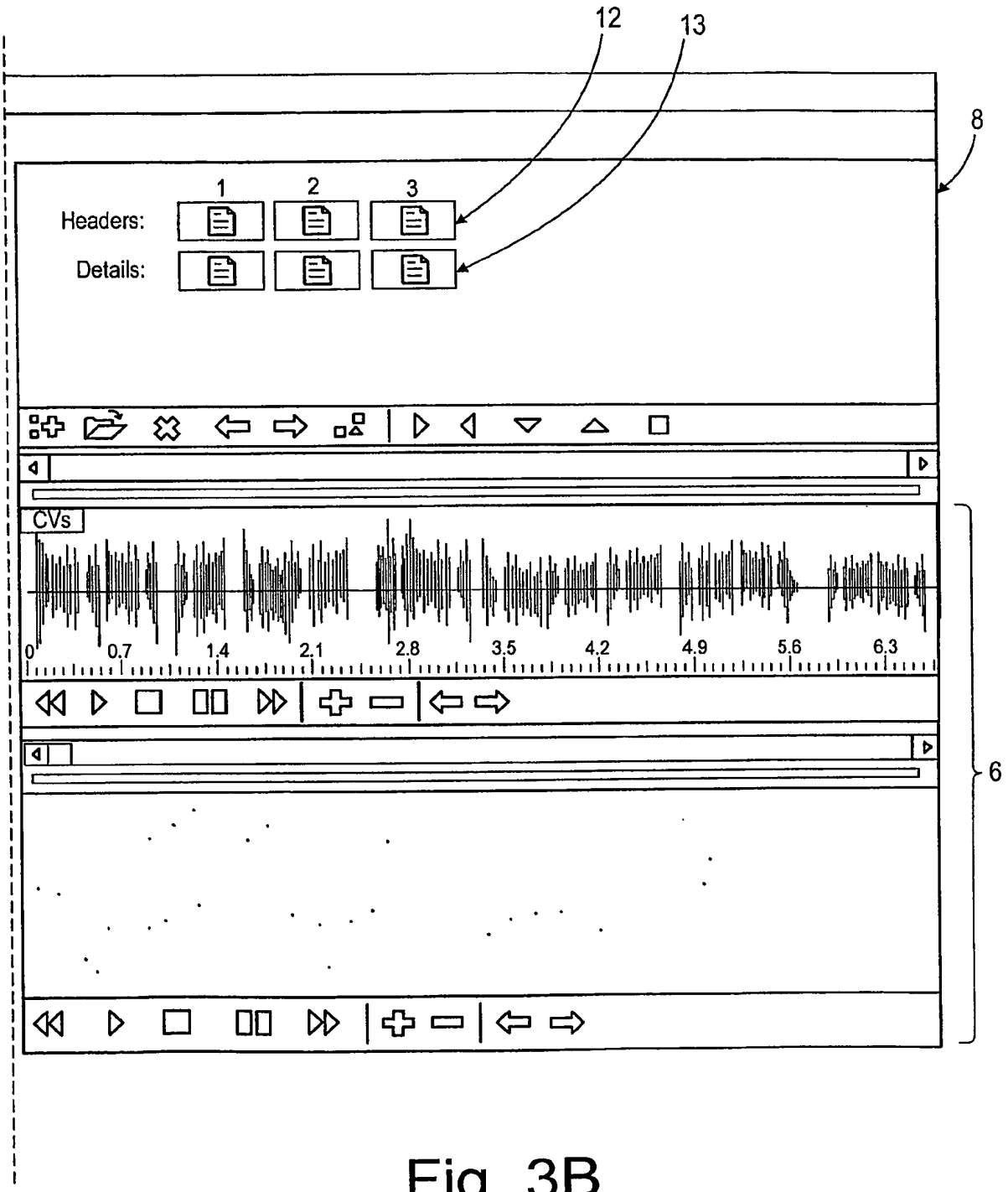


Fig. 3B

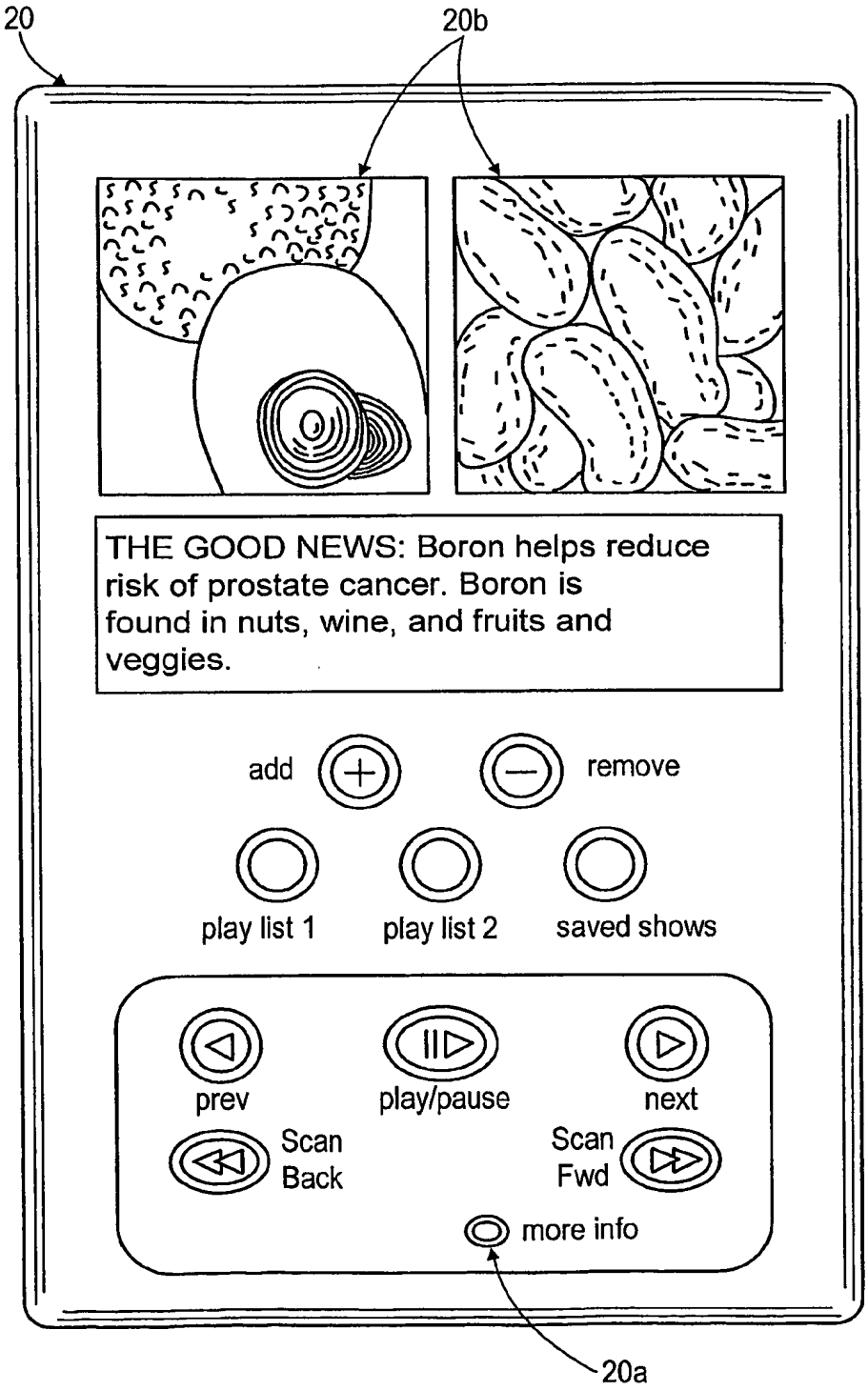

20
20b
THE GOOD NEWS: Boron helps reduce risk of prostate cancer. Boron is found in nuts, wine, and fruits and veggies.
add
remove
play list 1
play list 2
saved shows
prev
play/pause
next
Scan Back
Scan Fwd
more info
20a

Fig. 4

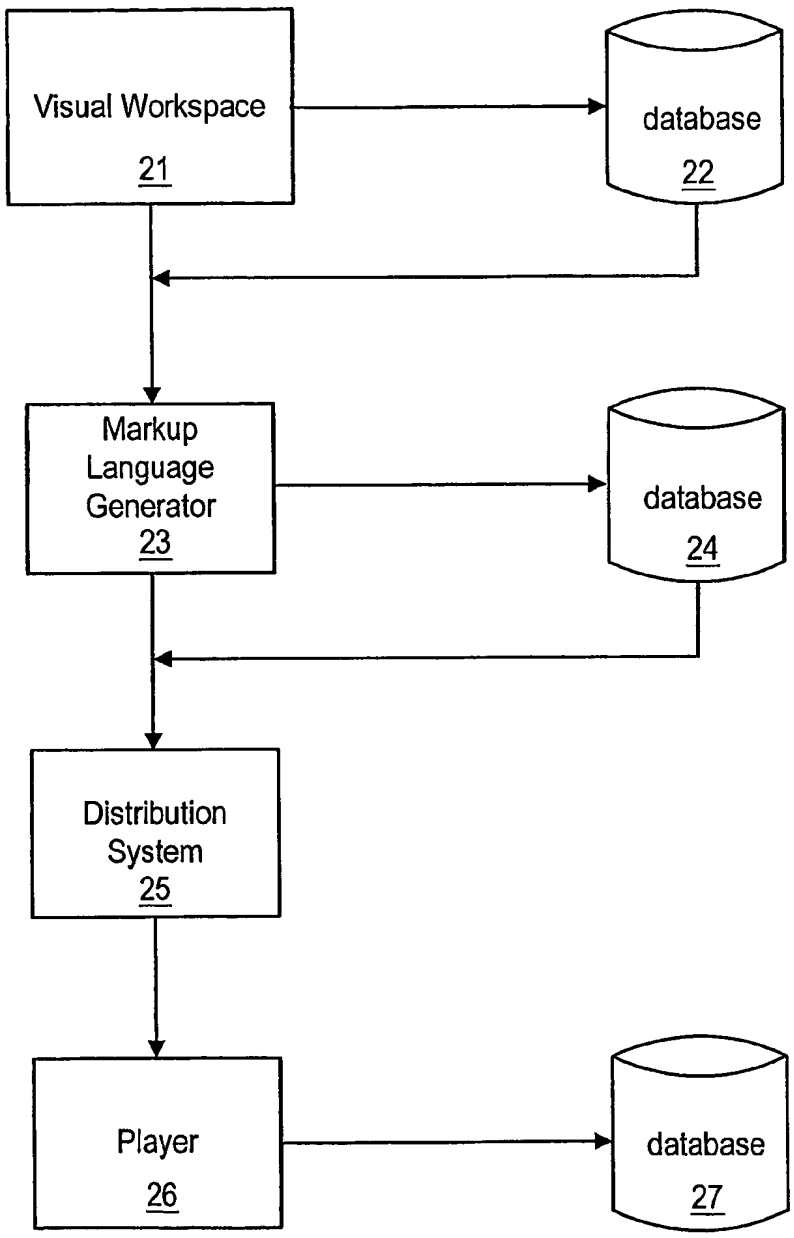

Visual Workspace
21
database
22
Markup
Language
Generator
23
database
24
Distribution
System
25
Player
26
database
27

Fig. 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US02/27820

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) :G06F 3/00, 15/62
US CL :345/719, 720, 721, 723, 727, 835; 707/501.1
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 345/717, 718, 719, 720, 721, 723, 726, 727, 731, 835; 707/500.1, 501.1, 512

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
NONE

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EAST

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 6,198,833 B1 [RANGAN et al] 06 March 2001, col. 4 lines 32-35, col. 5 lines 15-59, colum 6 lines 48-51, col. 8 lines 3-45, col. 9 lines 3-5, col. 10 lines 53-58, co. 11 lines 7-18, abstract, title.	1-21.
Y	US 5,237,648 A [MILLS et al] 17 August 1993, col. 2 lines 20-23, col. 3 lines 47-54, col. 4 lines 59-65, col. 6 lines 30-33, col. 8 lines 57-60, col. 9 lines 59-62.	1-21.
Y,P	US 6,357,042 B2 [SRINIVASAN et al] 12 March 2002, col. 6 lines 8-14, col. 6 lines 32-40, col. 8 lines 5-7, col. 8 lines 15-30, col. 11 lines 53-56, col. 29 lines 17-46, abstract.	1-21.

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search	Date of mailing of the international search report
30 SEPTEMBER 2002	13 NOV 2002

Name and mailing address of the ISA/US	Authorized officer
Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231	RAYMOND J. BAYER James R. Matthews
Facsimile No. (703) 305-3230	Telephone No. (703) 305-9789

Form PCT/ISA/210 (second sheet) (July 1998)★

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US02/27820

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y,P	US 6,424,361 B1 [CHAPUIS] 23 July 2002, col. 3 lines 28-29, col. 4 lines 51-54, col. 5 lines 4-18, col. 8 lines 27-32.	1-21.
A,P	US 6,366,914 B1 [STERN] 02 April 2002, see entire document.	1-21.